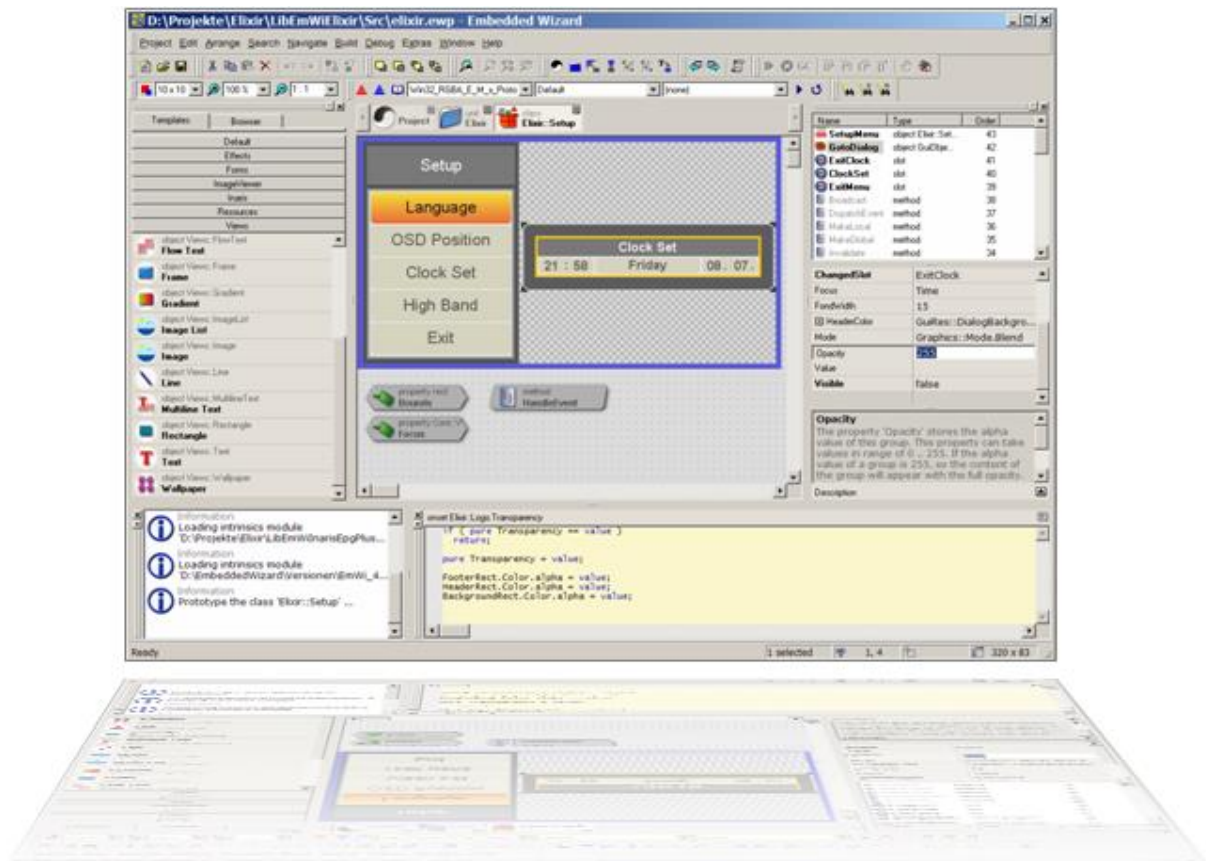




Embedded Wizard

IDE User Manual



Version 6.51
28 April 2015

Authors: Dipl. Ing. Paul Banach, Dipl. Ing. Manfred Schweyer
Copyright © TARA Systems GmbH

Contents

User Interface	3
1 Main Window	5
1.1 Project Menu	6
1.2 Edit Menu.....	6
1.3 Arrange Menu	7
1.4 Search Menu	8
1.5 Navigate Menu.....	9
1.6 Build Menu.....	10
1.7 Debug Menu	14
1.8 Extras Menu	15
1.9 Window Menu	16
1.10 Help Menu	18
2 Gallery	19
2.1 Templates view.....	19
2.2 Browser.....	21
3 Composer	25
3.1 Project Composer.....	28
3.2 Profile Composer.....	29
3.3 Unit Composer.....	30
3.4 Enum Composer.....	33
3.5 Set Composer.....	33
3.6 Class Composer	34
4 Inspector	37
4.1 Member Area.....	37
4.2 Attribute and Property Area	40
4.3 Description Area	45
5 Editor	46
5.1 Declaration Area.....	46
5.2 Implementation Area	48
6 Prototyper	51
7 Variables Window	53
8 Callstack Window.....	54
9 Chora Objects Window	55
10 Garbage Collector Window	56
11 Log Window	57

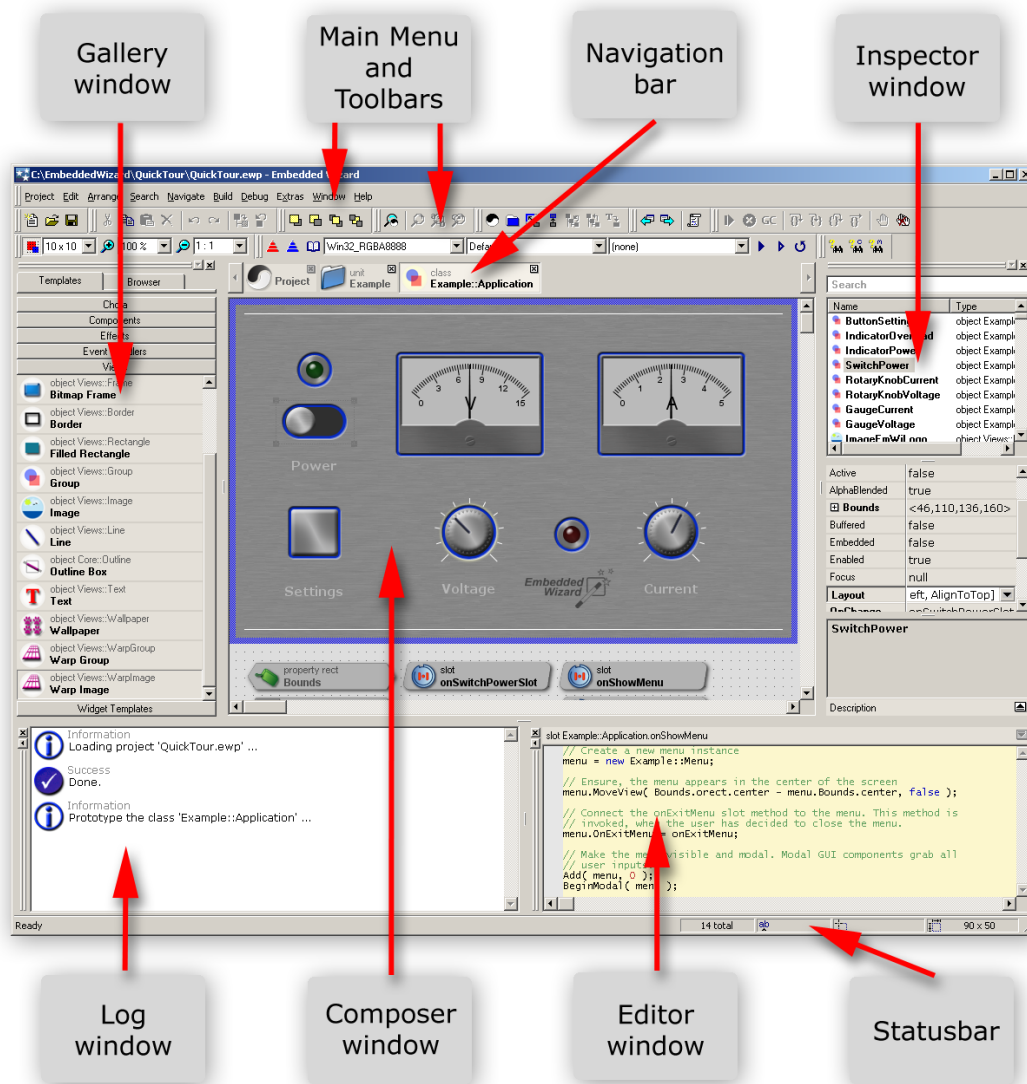
User Interface

The user interface of the Embedded Wizard is designed to create and modify GUI applications in a comfortable way. Complex GUIs can be build up by combining basic components without writing a single line of code. Only the implementation of methods requires direct source code editing.

Like other modern Windows applications, the Embedded Wizard supports several editing features:

- Drag & Drop – move software components between the different windows and applications with the mouse, e.g. you can take a template with the mouse from the Gallery and move it into the Composer.
- Copy & Paste – move or copy software components into the clipboard, e.g. you can copy an object from the Composer into the clipboard by pressing *Ctrl+C*.
- Undo & Redo – go back and forward in the edit history, e.g. you can restore a deleted object by pressing *Ctrl+Z*.
- Context Menu – display a popup menu containing the most important actions by pressing the right mouse button.

The Embedded Wizard user interface consists of the following windows and editors:

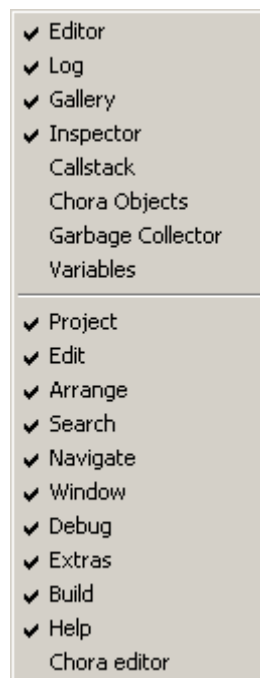


- [Main Window](#) (Chapter 1)
- [Gallery](#) (Chapter 2)
- [Composer](#) (Chapter 3)
- [Inspector](#) (Chapter 4)
- [Editor](#) (Chapter 5)
- [Prototyper](#) (Chapter 6)
- [Variables Window](#) (Chapter 7)
- [Callstack Window](#) (Chapter 8)
- [Chora Objects Window](#) (Chapter 9)
- [Garbage Collector Window](#) (Chapter 10)
- [Log Window](#) (Chapter 11)

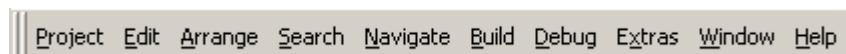
1 Main Window

The main window of the Embedded Wizard contains a menu bar, several toolbars and the workspace for other windows and the Composer. The menu, the toolbar the Gallery, the Inspector, the Editor and the Log Window are dockable, this means, that each window (or toolbar) can either be integrated into the main window or it can be displayed as a standalone floating window. Just double click on the gripper or on the caption of the affected dockable window to toggle between these two display modes.

When you click with the right mouse button onto the menu bar or onto the toolbar, a context menu appears where you can hide and show the different toolbars and windows. The following image shows the context menu:



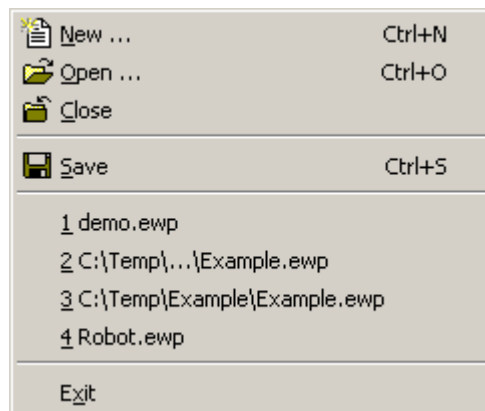
The menu bar of the main window contains the following menu items:



- [Project Menu](#) (Chapter 1.1)
- [Edit Menu](#) (Chapter 1.2)
- [Arrange Menu](#) (Chapter 1.3)
- [Search Menu](#) (Chapter 1.4)
- [Navigate Menu](#) (Chapter 1.5)
- [Build Menu](#) (Chapter 1.6)
- [Debug Menu](#) (Chapter 1.7)
- [Extras Menu](#) (Chapter 1.8)
- [Window Menu](#) (Chapter 1.9)
- [Help Menu](#) (Chapter 1.10)

1.1 Project Menu

The *'Project'* menu contains general commands for creating, loading and saving an Embedded Wizard project:



- *'New...'* – Create a new Embedded Wizard project.
- *'Open...'* – Open an already existing Embedded Wizard project.
- *'Close'* – Close the currently opened Embedded Wizard project.
- *'Save'* – Save the currently opened Embedded Wizard project.
- *'Exit'* – Quit the Embedded Wizard application.

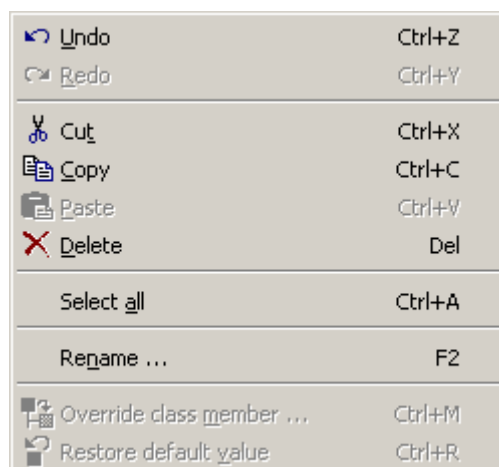
Additionally, the *'Project'* menu contains a list with the recently used projects.

These commands are also available in the *'Project'* toolbar:



1.2 Edit Menu

The *'Edit'* menu contains commands for Undo/Redo, for controlling the clipboard and for renaming the currently selected members:



- *'Undo'* – Cancel the last modification. Each selection of *'Undo'* goes one step back through the history of modifications.

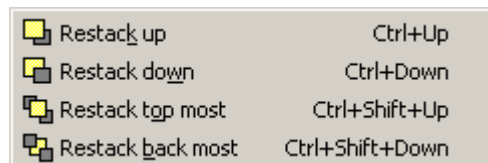
- *'Redo'* – Cancel the last *'Undo'* operation. Each selection of *'Redo'* goes one step ahead through the history of modifications.
- *'Cut'* – Cut the currently selected members or text and put it on the clipboard.
- *'Copy'* – Put a copy of the currently selected members or text on the clipboard.
- *'Paste'* – Insert the current clipboard contents.
- *'Delete'* – Delete all currently selected members or text.
- *'Select all'* – Select all members or the entire text in the Editor.
- *'Rename ...'* – Rename the currently selected member.
- *'Override class member'* – This command is used to override an inherited class member. Thereupon the member can be modified. To return to the origin, inherited member, delete the overridden member.
- *'Restore default value'* – Restores the inherited value of an attribute or property. In derived members, the values of attributes and properties can be overridden. To restore the origin, inherited value of such attribute or property use this command.

These commands are also available in the *'Edit'* toolbar:



1.3 Arrange Menu

The *'Arrange'* menu contains commands for restacking the currently selected members:



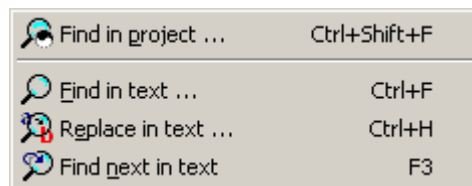
- *'Restack up'* – Restack all currently selected members up (increase Z-order).
- *'Restack down'* – Restack all currently selected members down (decrease Z-order).
- *'Restack top most'* – Restack all currently selected members topmost (set topmost Z-order).
- *'Restack back most'* – Restack all currently selected members backmost (set backmost Z-order).

These commands are also available in the *'Arrange'* toolbar:



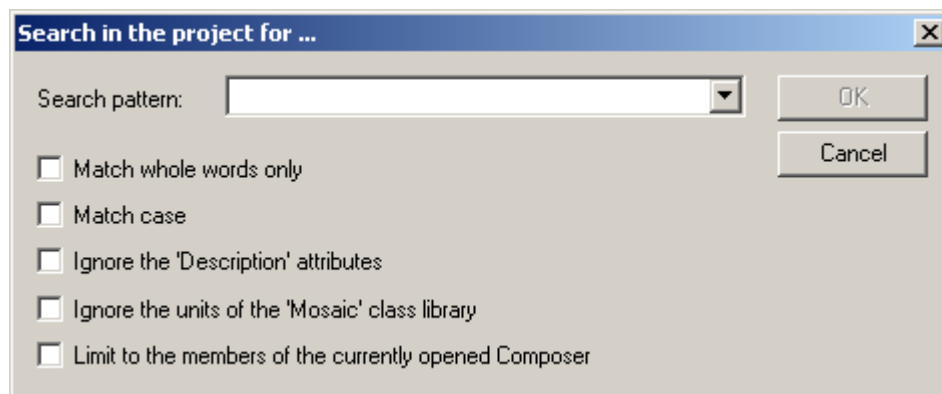
1.4 Search Menu

The 'Search' menu contains commands for searching members within the project and for searching or replacing text within the Editor.



- 'Find in project...' – Search in the project for members containing the given text. All found locations are reported in the Log Window. A double click onto a message will guide you to the location.

When selecting this menu item, a small dialog box appears, where the search text can be inserted and search options can be defined.



Please note the option 'Limit to members of the currently opened Composer'. This option limits the scope of search to the content of the currently opened Composer page.

- 'Find...' – Search in the Editor ("Editor" (Chapter 5)) for a given text. When selecting this menu item, a small dialog box appears, where the search text can be inserted and search options can be defined.

The 'Find Next' button moves the caret in the Editor to the next location relative to the current caret position.

- 'Replace...' – Search in the Editor for a given text and replace it by a new term. The command displays a dialog to insert the search term, the substitution term and some search options.

The 'Find Next' button moves the caret in the Editor to the next location relative to the current caret position without substitution.

The 'Replace' button replaces the next found term.

The 'Replace All' button substitutes all search terms without any further confirmation.

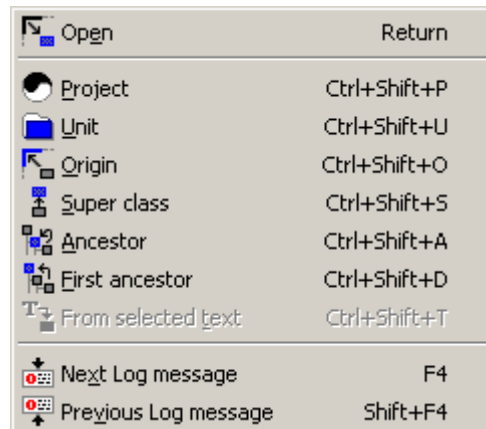
- 'Find next' - Repeat the search operation in order to move the caret to the next occurrence of the search text in the Editor. You can simply press *F3* to jump the caret to the next occurrence of the search text.

These commands are also available in the ‘Search’ toolbar:



1.5 Navigate Menu

The ‘*Navigate*’ menu contains helpful commands to navigate quickly inside a project. These commands are context sensitive – they depend on the currently edited content or the selected members.



- ‘*Open*’ – Open the selected member or its type definition. After selecting this command, the content of the selected member is shown in the Composer (“Composer” (Chapter 3)) or Editor (“Editor” (Chapter 5)). The content of a member can be also opened by a double click.
- ‘*Project*’ – Navigate to the project. After selecting this command, the project page is shown in the Composer (“Composer” (Chapter 3)).
- ‘*Unit*’ – Navigate to the unit, where the currently edited class or enumeration is defined in. After selecting this command, the unit is displayed in the Composer.
- ‘*Origin*’ – Navigate to the location, where the currently edited content is defined. After selecting this command, the Composer is opened, where the brick of the edited content is defined in.
- ‘*Super class*’ – Navigate to the super class of the currently edited class. After selecting this command, the content of the super class is displayed in the Composer.
- ‘*Ancestor*’ – Navigate to the ancestor of the selected or edited member. This command is useful by derived members only. The location of the immediate ancestor is shown in the Composer.
- ‘*First ancestor*’ – Navigate to the first ancestor of the selected or edited member. This command is useful by derived members only. The location of the origin (the oldest) ancestor is shown in the Composer.
- ‘*From selected text*’ – Navigate to the member, whose name is selected in the text field of the Editor (“Editor” (Chapter 5)) or the Inspector window (“Inspector” (Chapter 4)).

- 'Next Log message' – Inspects the next Log window message. ("Log Window" (Chapter 11))
- 'Previous Log message' – Inspects the previous Log window message. ("Log Window" (Chapter 11))

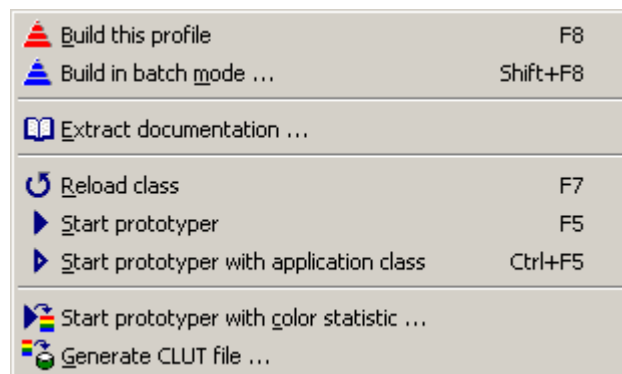
These commands are also available in the 'Navigate' toolbar:



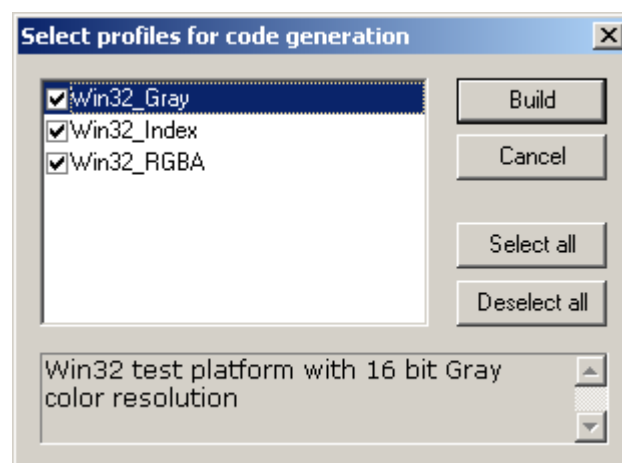
Within the Editor, the navigation commands 'Origin', 'Ancestor' and 'First ancestor' can be used to find the location, where the method brick is defined and to inspect the base version of a derived method.

1.6 Build Menu

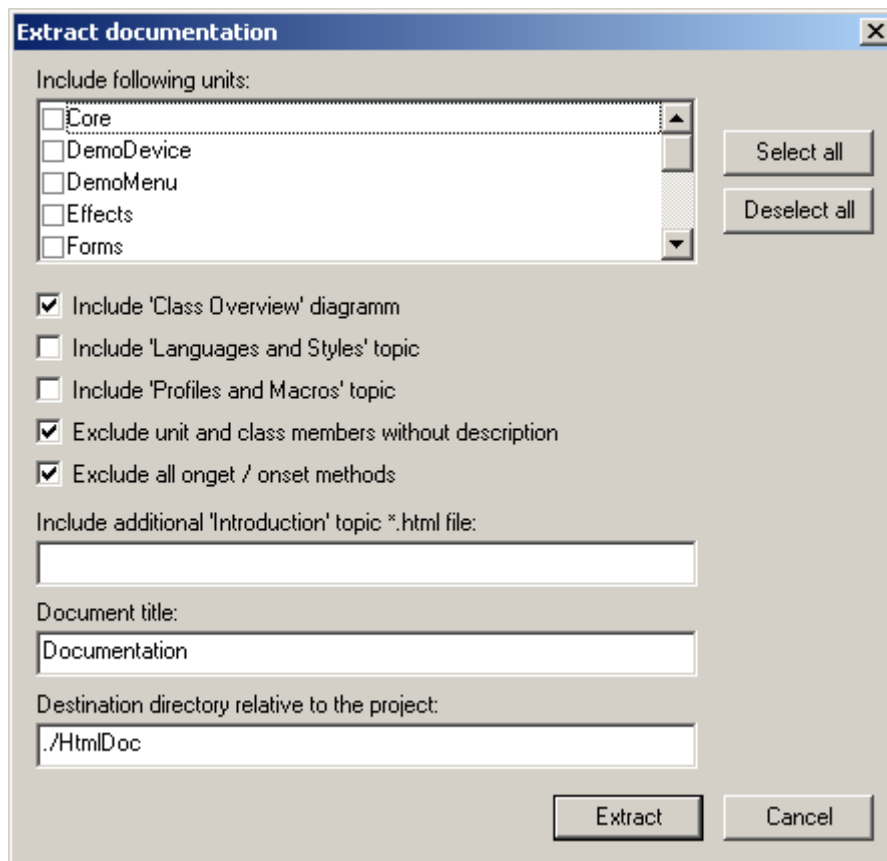
The 'Build' menu contains commands to start the code generation and to control the Prototyper:



- 'Build this profile' – Start the code generation process for the currently selected profile.
- 'Build in batch mode ...' – Display a dialog box to select one or more profiles and start the code generation process for all selected profiles.



- 'Extract documentation ...' – Display the following dialog to choose the options for the generated documentation:



The upper part of the dialog contains a list of all units of the currently loaded project. Each unit can be included or excluded from the documentation generation.

The center part of the dialog contains a set of check boxes, to generate an additional class overview diagram, to include the description of all languages, profiles or macros and to exclude some superfluous documentation.

The lower part of the dialog contains three edit fields to include some external introduction documentation (as an external HTML file), to define the document title and to select an output directory.

After starting the process by pressing 'Extract', Embedded Wizard evaluates the content of your project, the dependencies between the classes, constants, resources, etc. and stores it together with the associated inline documentation within a set of HTML files. In a second step a single Microsoft HTML Help file is created. This help file contains detailed documentation of your project and it can be used or distributed as your 'software reference manual'.

For more details about the documentation features of Embedded Wizard see "Chora User manual").

- 'Reload class' – Reload the currently edited class once again into the Composer and reinitialize it. Sometimes, if the implementation of a method has been changed, new bricks are added to, some bricks are removed from the class, or members are renamed you have to reload the class in order to refresh the Chora interpreter.
- 'Start prototyper' – Open the Prototyper window ("Prototyper" (Chapter 6)) and load the currently edited or in Composer selected class into the Prototyper.

- *'Start prototyper with application class'* – Open the Prototyper window (*"Prototyper"* (Chapter 6)) and load the application class into the Prototyper.

The name of the application class has to be defined before in the attribute `ApplicationClass` within the current profile. For more details about profiles see *'Chora User Manual'*.

- *'Start prototyper with color statistic ...'* – Open the Prototyper window (*"Prototyper"* (Chapter 6)) and load the currently edited or in Composer selected class into the Prototyper. During the prototyping session, a statistic about all used colors of the prototyped GUI application is generated and stored as a text file within the project directory.

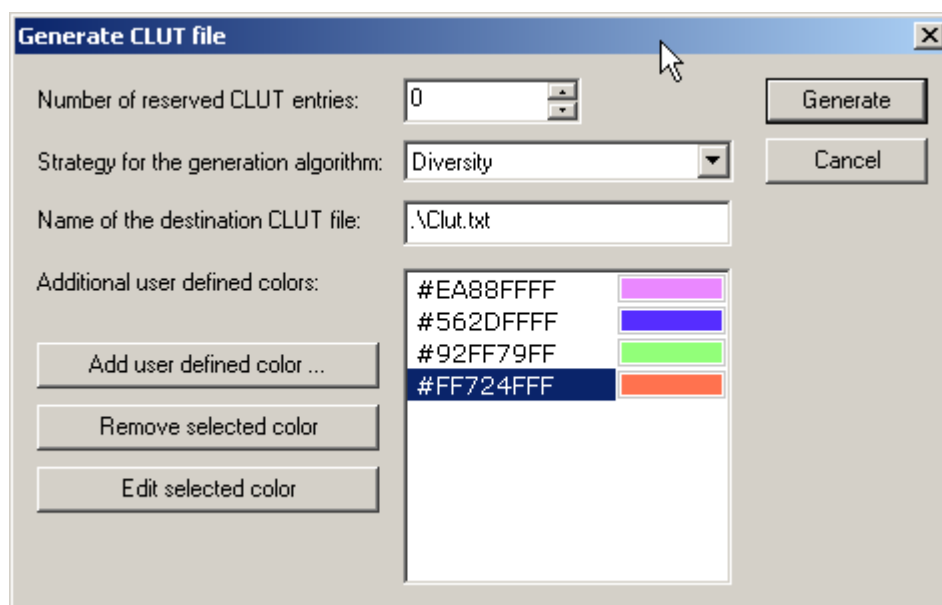
The statistic file contains all used color values and their number of usage. This color statistic is used as input for the CLUT generation (see below). The prototyping session with color statistic can be seen as a learning phase for the CLUT generation: the more GUI screens are displayed during the prototyping session, the more color values are added to the color statistic and the better results can be created from the CLUT generator.

If there is already an existing color statistic file available, a dialog appears at the beginning of the prototyping session: the user can either add all colors of the new prototyping session by selecting *'Resume'* or create a new color statistic by selecting *'Discard'*.

Please note, to run the color statistic you have to select a profile, which is using the platform package *'Tara.Win32.RGBA8888'*.

- *'Generate CLUT file ...'* – Convert a color statistic file into a color-lookup-table (CLUT). Generally, the CLUT generator converts a color statistic (see above) of the current project into a CLUT file containing the 256 best fitting colors. This CLUT file can be used later within an Index8 Platform Package as a global color palette.

After selecting this menu item, the following dialog appears:



Within the first line, the number of reserved CLUT entries can be defined. Reserved CLUT entries are only used for external applications (so called Applets) sharing the CLUT with the Embedded Wizard GUI application.

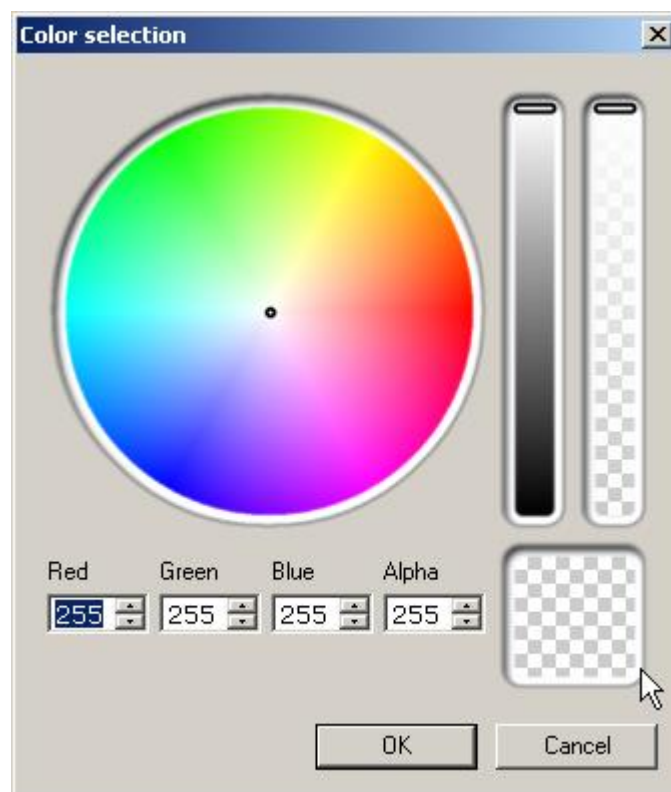
Within the second line of the dialog, the strategy for CLUT generation can be selected:

- The *'Majority'* strategy takes care on colors with a high occurrence, whereas colors with low occurrence are merged.
- The *'Diversity'* strategy takes care that all different colors are kept within the CLUT, independent from their real occurrence.
- The *'Popularity'* strategy takes care on all different colors, which are often used, and combines very different colors only if they are used rarely. This strategy can be seen as a compromise between Majority and Diversity.

Within the third line of the dialog, the name of the resulting CLUT file can be defined. The given path is relative to the current project directory.

Within the lower part of the dialog, user defined colors can be given to the CLUT generator. These colors are included within the final CLUT, independent from their occurrence within the color statistic.

User defined colors can be selected comfortable by the following dialog, which appears after selecting the buttons *'Add user defined color ...'* or *'Edit selected color ...'*:



User defined colors may be useful for external applications (Applets), which expects a CLUT with a set of fixed colors. Normally, the CLUT generation should be based only on the color statistic file.

The CLUT generation is started by pressing the button 'Generate'. This process may take between a few seconds and several minutes. The number of colors, which are reduced by the CLUT generator are reported within the Log window.

Additional remark: In order to use the generated CLUT within your Index8 based platform, the CLUT file has to be specified in the Profile. For this purpose enter the name of the CLUT file in the attribute Clut of the affected profile.

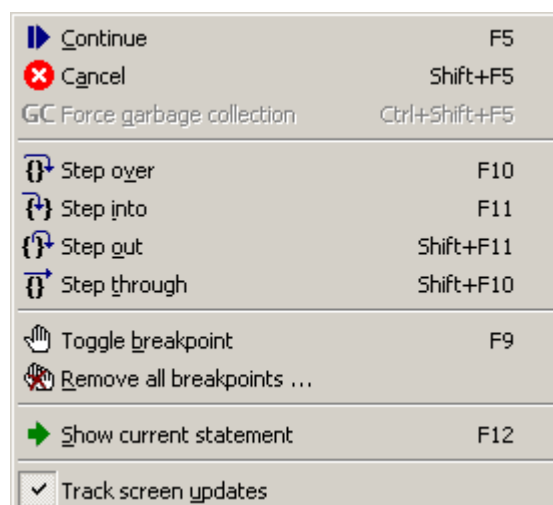
These commands and additional combo boxes are available in the 'Build' toolbar:



- 'Profile' combo box – In the first combo box the profile used for prototyping purposes and for code generation can be selected. The selected profile is used when a class is loaded into the Prototyper or when the code generation process is started by selecting the menu item 'Build this profile'.
- 'Language' combo box – In the second combo box the active language for prototyping purposes can be selected. The currently active language determines the language dependent variant of a resource or constant. This selection can be compared with the Chora global variable `language`, which changes the currently active language at the runtime. See "Chora User Manual".
- 'Styles' combo box – In the third combo box the set of active styles for prototyping purposes can be selected. The active styles determine the used variants of multivariant classes, constants and resources. This selection can be compared with the Chora global variable `styles`, which changes the active styles at the runtime. See "Chora User Manual".

1.7 Debug Menu

The 'Debug' menu contains commands for debugging GUI applications:

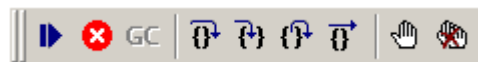


- 'Continue' – Continue the code execution in the Prototyper.

- *'Cancel'* – Cancel the execution of the current event or timer expiration and continue with the next event. The Prototyper window still keeps opened.
- *'Force garbage collection'* – Start immediately the next garbage collection cycle.
- *'Step over'* – Step over the next statement.
- *'Step into'* – Step into the statement. In case of a method call, step into the method.
- *'Step out'* – Execute code until the currently debugged method is left.
- *'Step through'* – Like *'Step over'*, but don't break at initialization expressions, defined in the Inspector (see ["Inspector"](#) (Chapter 4)). Execute until the next statement in a method has been reached.
- *'Toggle breakpoint'* – Set or remove the breakpoint at the current text row in the Editor (see ["Editor"](#) (Chapter 5))
- *'Remove all breakpoints ...'* – Allows you to remove all breakpoints at once.
- *'Show current statement'* – Shows the location where the execution is halted in the Editor or Inspector.
- *'Track screen updates'* – Determines whether the Prototyper should highlight the redrawn screen areas. This function can help you to analyze and optimize the screen updates in your GUI application.

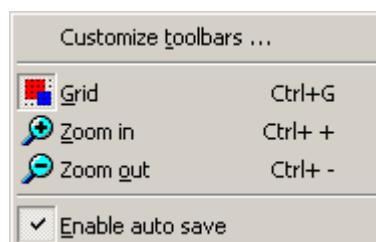
If enabled, the Prototyper masks the redrawn screen areas with green color, which after a short period is faded out again.

These commands are also available in the *'Debug'* toolbar:



1.8 Extras Menu

The *'Extras'* menu contains helpful commands to customize the appearance and to set a grid and zoom factor in the Composer (see ["Composer"](#) (Chapter 3)).



- *'Customize toolbars ...'* – Display a dialog box to configure the toolbars. Buttons can be added to a toolbar by dragging the button from the dialog box to the toolbar. Buttons can be removed from a toolbar by dragging the button from the toolbar back to the dialog box.
- *'Grid'* – Enable or disable the grid in the Composer. When a grid is activated, all objects and bricks, which are moved with the mouse, are aligned to the grid points.

- 'Zoom in' – Increases the zoom factor of the currently active page of the Composer.
- 'Zoom out' – Decreases the zoom factor of the currently active page of the Composer.
- 'Enable auto save' – Determines whether Embedded Wizard should save periodically the last project modifications. If enabled, Embedded Wizard performs every 20 minutes the usual project save operation as if it would be triggered by the menu item 'Save'.

Please note, like the usual save operation, auto save overwrites your project files if there are changes in the project only.

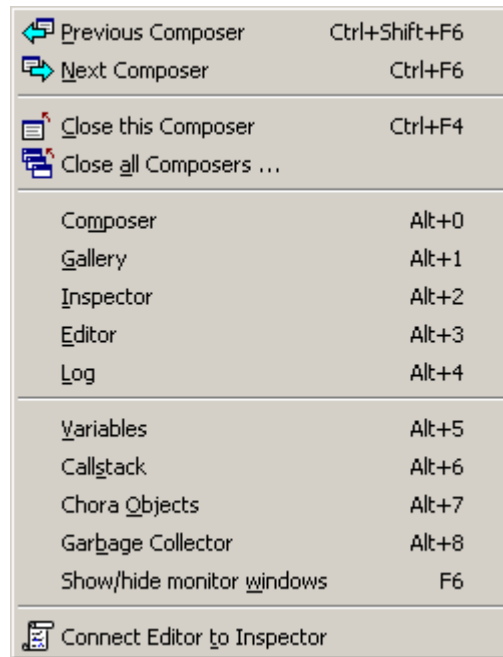
These commands and three additional combo boxes are available in the 'Extras' toolbar:



- 'Grid' combo box – Select the size of the grid for the currently active Composer page. The grid is displayed as gray points in the background of the Composer window.
- 'Zoom' combo box – Select the zoom factor for the currently active Composer page.
- 'Aspect Ratio' combo box – Change the aspect ratio of the currently active Composer page. The combo box contains a list of predefined aspect ratios. Additionally a user-defined aspect ratio can be specified. In this case you have to input both, the horizontal and the vertical expansion divided by a colon sign : directly in the edit field of the combo box.

1.9 Window Menu

The 'Window' menu contains commands to navigate through the Composer pages and the dockables of the Embedded Wizard:



- 'Previous Composer' - Navigate to the previous Composer page (see "[Composer](#)" (Chapter 3)).
- 'Next Composer' - Navigate to the next Composer page.
- 'Close this Composer' – Close the currently active Composer page.
- 'Close all Composers ...' – Close all Composer pages excepting the project page (which is always opened as long as a project is loaded).
- 'Composer' – Activate the Composer window.
- 'Gallery' – Shows and activates the Gallery window (see "[Gallery](#)" (Chapter 2)).
- 'Inspector' – Shows and activates the Inspector window (see "[Inspector](#)" (Chapter 4)).
- 'Editor' – Shows and activates the Editor (see "[Editor](#)" (Chapter 5)).
- 'Message Log' – Shows and activates the Log window (see "[Log Window](#)" (Chapter 11)).
- 'Variables' – Shows and activates the Variables monitor window (see "[Variables Window](#)" (Chapter 7)).
- 'Callstack' – Shows and activates the Callstack monitor window (see "[Callstack Window](#)" (Chapter 8)).
- 'Chora Objects' – Shows and activates the Chora Objects monitor window (see "[Chora Objects Window](#)" (Chapter 9)).
- 'Garbage Collector' – Shows and activates the Garbage Collector window (see "[Garbage Collector Window](#)" (Chapter 10)).
- 'Show/hide monitor windows' – Hides temporarily the monitor windows and restores their state again.

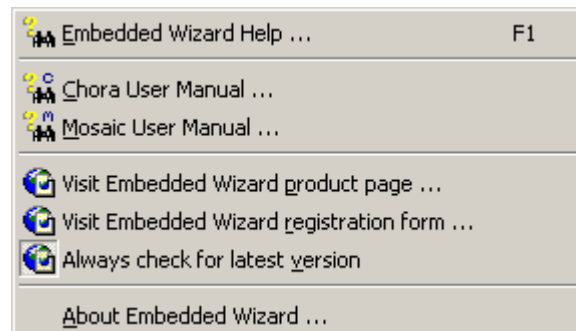
- *'Connect Editor to Inspector'* – Connect the Inspector with the Editor. If the option is active, the Editor shows the content of attributes and properties selected in the Inspector.

Some of these commands are also available in the *'Window'* toolbar:



1.10 Help Menu

The *'Help'* menu contains commands for displaying the online documentation and the version information:



- *'Embedded Wizard Help ...'* – Display this Embedded Wizard User Manual.
- *'Chora User Manual ...'* – Display the Chora User Manual.
- *'Mosaic User Manual ...'* – Display the Mosaic User Manual.
- *'Visit Embedded Wizard product page ...'* – This command guides you to the Embedded Wizard webpage, where updates, documentations and examples can be found.
- *'Visit Embedded Wizard registration form ...'* – This command guides you to a registration webpage. We recommend you to register your Embedded Wizard license in order to be able to receive updates and news.
- *'Always check for latest version'* – This command determines, whether Embedded Wizard automatically checks for new updates.
- *'About Embedded Wizard ...'* – Display a window with version and copyright information.

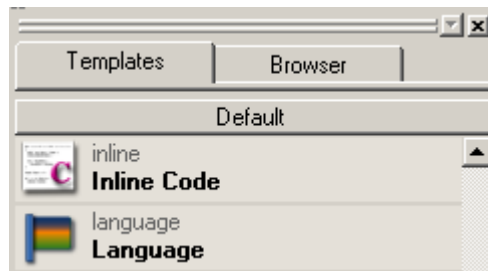
These commands are also available in the *'Help'* toolbar:

2 Gallery

The Gallery of the Embedded Wizard accommodates two different views:

- Templates view (see “[Templates view](#)” (Chapter 2.1))
- Browser view (see “[Browser](#)” (chapter 2.2))

To switch between the different views, the Gallery window provides a tab control. To switch to the Browser view, click on the *Browser* tab. To switch to the Templates view, click on the *Templates* tab:

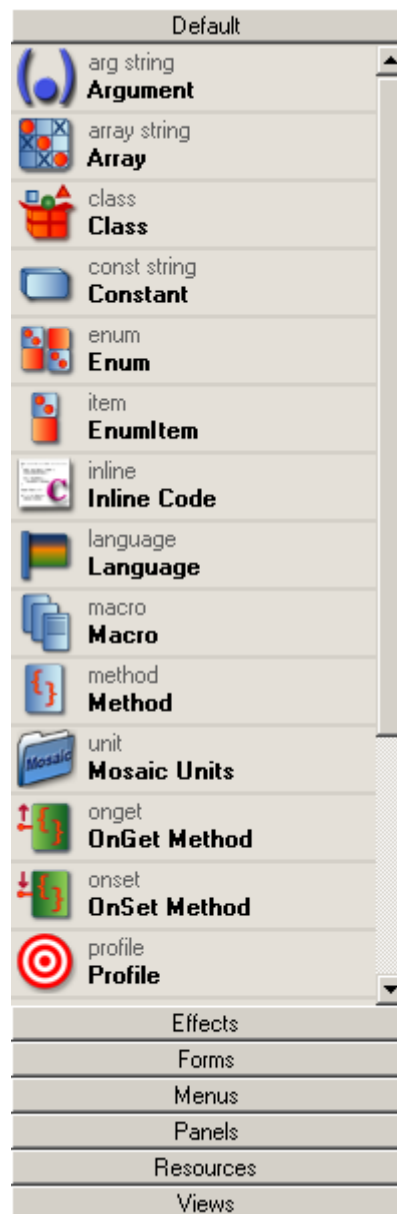


As long as the Gallery is the active window, you can also press the keys *Ctrl+Tab* to toggle between these two views.

2.1 Templates view

The Templates view can be regarded as a pool of templates, which can be used to build up your GUI application. Any kind of template may be stored in the Gallery, e.g. a simple variable, a complex method or the implementation of a complete class.

The Templates view is organized in folders, which can be opened by a single click on the caption (on the bar) of the folder. As soon as a project is loaded into Embedded Wizard, a set of default folders is loaded into the Templates view:



The templates can be dragged from the Gallery and dropped into the Composer in order to use them. You can also copy a selected template to the clipboard by pressing **Ctrl+C** and insert it into the Composer by pressing **Ctrl+V**.

Additionally, the Templates view can be used to store your own, frequently used templates. For example, a member selected in the Composer can be copied by **Ctrl+C** to the clipboard and then be pasted by **Ctrl+V** into an opened folder of the Templates view.

You can also drag selected members from the Composer by holding down the **Ctrl** key and drop them into the Templates view. In the same manner, you can copy or move templates between the several folders of the Templates view.

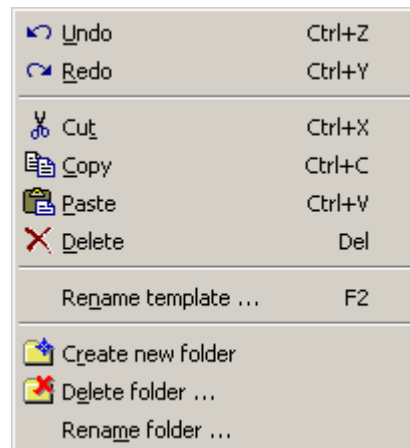
As long as the Templates view is active, you can press a key in order to open a folder, whose name starts with the pressed key. For example, if you want to open the folder *Panels*, press the key **P**. If there are more folders, which start with the sign **P**, press the key **P** more times until the desired folder has been reached.

In the same manner you can select a template within the opened folder. In order to distinguish between the shortcuts for the folder selection and the shortcuts for the template selection, the templates selection has always to start with the *Space* key. For example, you wish to select a *Push Button* template in the currently opened folder. In this case press one after another the keys *Space* and *P*. If there are more templates, which starts with P, press the key *P* more times until the desired template has been reached.

By using these shortcuts you can copy templates very quickly without the necessity to click with the mouse on them. For example to copy and paste a method from the Default folder into the Composer use following shortcuts:

First press *Alt+1* to switch to the Gallery. Then press *D* to open the Default folder. Then press *Space* key followed by *M M* to select the method template. With *Ctrl+C* copy the method template and with *Ctrl+O* switch to the Composer where the copied template can be pasted via *Ctrl+V*.

When clicking with the right mouse button into the Templates view, the following context menu appears, which provides commands to create, delete and rename folders and templates:

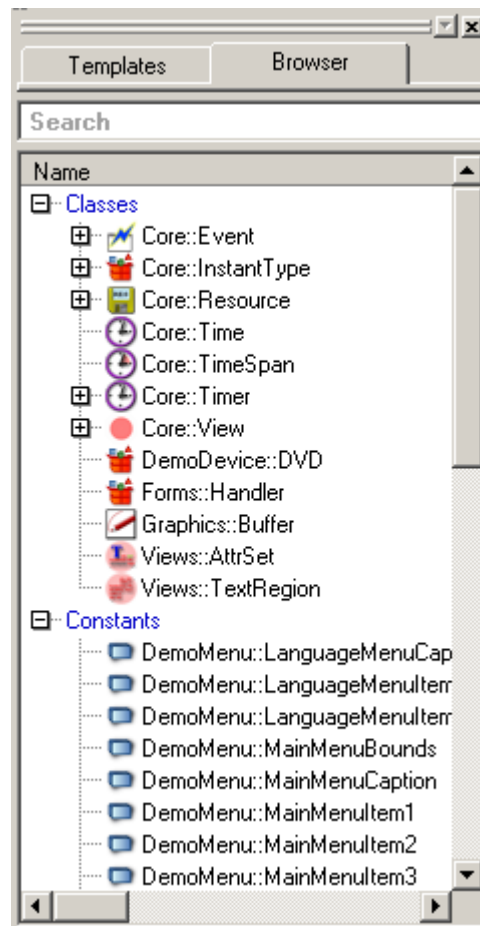


All folders within the Templates view are mapped to a subdirectory of the Embedded Wizard installation. All templates are stored as readable Chora code within these subdirectories. If you want to transfer some of your template folders to another PC, you have to copy the subdirectories into the other Embedded Wizard installation and to restart Embedded Wizard.

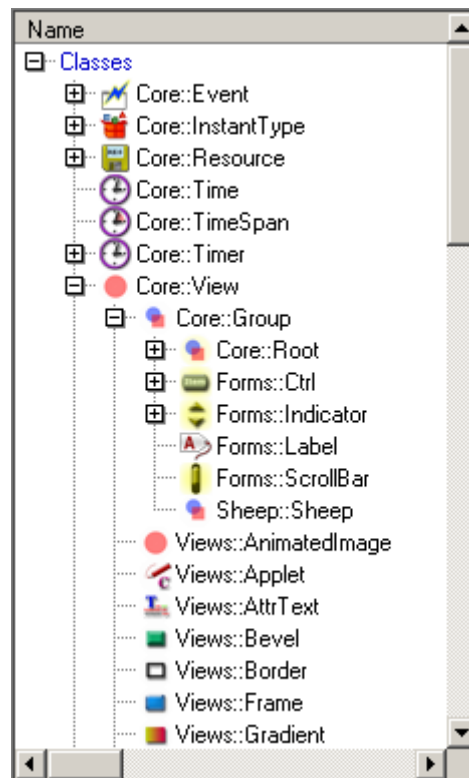
To avoid the deletion of a template by mistake, you can set your template files to write protected.

2.2 Browser

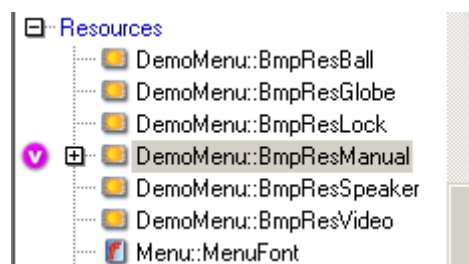
The Browser is the second view of the Gallery window. It displays a list of project members and gives you an overview about the project structure:



The Browser view is divided in 2 parts. In the bottom area a list of project members is displayed. The members are arranged in six categories '*Classes*', '*Constants*', '*Resources*', '*Enumerations*', '*Auto objects*' and '*Sets*'. The organization of the displayed items within the Browser does reflect the inheritance hierarchy of the corresponding members. In order to show all descends of a member simply click on the small  icon. The click on the  icon will collapse the member, so its descends are hidden:



The multi-variant members are handled in a special manner by the Browser view. They are always signed with the icon  in the first column. This sign should help you to recognize multi-variant members at a glance. The concept of variants allows the development of GUI applications, whose appearance and behavior can be changed at the runtime. This unique feature simplifies the implementation of skins, themes, target and resolution dependent GUIs, etc. This feature also provides the customization of existing software in a way very transparent to the customer. When a class, constant or resource has been overridden by a variant, it becomes multi-variant and will be signed with this small icon:



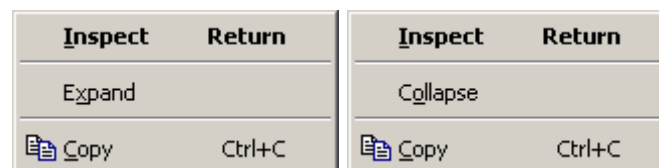
For details concerning the variants, please see the Chora User Manual.

Unlike the Composer or Inspector window, the Browser does not provide any modifications on the displayed members. Anyway, the Browser can serve as the source for all drag&drop operations. It is not necessary to switch between the Composers, if you need to drag a member from one Composer to another. Drag the member directly from the Browser instead. In this manner you can create a copy of a member, you can derive a sub-class or a variant of a member, or a new object of a class can be embedded into another class.

To copy a member, drag it by holding the *Ctrl* key down. To derive a new sub-class from another class, drag the base class member by holding the *Ctrl+Shift* keys down into a unit Composer. To derive a variant from a class, resource or constant, the keys *Ctrl+Shift+Alt* should be hold down. To derive an object of a class, the keys *Ctrl+Alt* should be hold down.

The members displayed in the Browser can also be inspected. For this purpose double click on the member or press *Return* key after a member has been selected. Thereupon the location, where the member is defined is shown in the Composer and the member can be edited as described in “[Composer](#)” (Chapter 3).

The Browser provides a context menu. Depending on the affected member, following menus do appear after a right mouse button click:



The upper area of the Browser provides a ‘Search’ edit field. This edit field can take a filter condition. If the filter is set, the Browser will highlight all members, whose name does fit the given filter condition:



The condition is fulfilled, if the filter text does appear anywhere within the members name. The filter evaluation is case insensitive. For example, if the filter is set to menu, the members `DemoMenu::MainMenu`, `Menu::MenuItem`, etc. are highlighted.

Optionally, parts of the filter can be specified using the wildcards '*' and '?'. The sign '*' stands for an arbitrary number of any signs. The '?' wildcard, on the other hand, stands for exactly one sign. When using wildcards, the entire name of the member has to fit the filter condition otherwise it will not become highlighted. To filter all members, set the filter to a single * sign.

To deactivate the filter click on the icon  on the right hand side of the edit field or press the *Esc* key as long as the edit field is focused.



As long as the Gallery is the active window, you can press the key *Tab* to toggle between the ‘Search’ edit field and the list of project members.

3 Composer

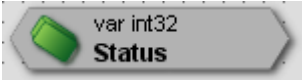
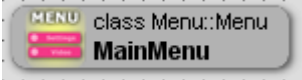
The Composer of the Embedded Wizard is used to edit GUI applications in a visual way. All graphical objects (like filled rectangles, images or texts) are directly drawn into the Composer. Other ingredients of a project are visualized as 'bricks'. A brick is a cover for a single Chora definition. This means, that each variable, method, property, class, ..., macro or unit of an GUI application is represented by a brick.



Each brick contains an icon, the type description and the name of the Chora definition. A brick can be selected, resized and moved with the mouse. The position of the bricks is not important for the functionality of your application; however, it is recommended to arrange all bricks clearly to keep track of the complete project. Sometimes, bricks overlap each other – this can be detected easily by the shadow of the bricks: The shadow of a brick is darker if another brick is lying behind.

The displayed icon and the shape of the brick reflect the role of the corresponding software component within the GUI application. Depending on the shape of the brick, the bricks can be divided in following three categories:

	Bricks with beveled sides define entities of logic. They are used for methods, slots, etc. The methods can be edited in the Editor. To edit the method double click on the brick with the left mouse button.
---	---

	Bricks with angle sides define entities of data. They are used for variables, properties, arrays, embedded objects, constants and resources. To initialize these entities, the desired initialization expressions can be specified in the Inspector.
	Bricks with rectangular shape stands for all others entities like units, classes, etc.

Per default all bricks are displayed with black text color. Only class members are signed with different text colors in order to reflect their origin and the class dependencies.

	All new defined, not inherited members are signed with the black text color.
	Inherited and overridden members will be signed with the gray text color.
	In case of errors in the inheritance of the particular member, its brick's name is drawn in red color. Such errors are detailed reported in the Log Window.

Since each project is organized in a hierarchical way (a project consists of units – a unit consists of classes - a class consists of variables, properties, methods, ...) the Composer is used to navigate through a project. A double click on a brick ‘zooms’ into the brick, e.g. a double click on an unit brick opens a new Composer page where all ingredients of the unit are shown. A double click on a method brick guides you to the Editor where the logic of the method can be implemented (see “[Editor](#)” (Chapter 5)).

To switch between the Composer pages, you can either click on one of the tabs inside the Navigation bar on top of the Composer or you can use the ‘*Previous Composer*’ / ‘*Next Composer*’ commands of the ‘*Window*’ main menu. To switch much more faster between the different Composer pages, you can use the following keys: *Ctrl+F6* switches to the next page, *Ctrl+Shift+F6* switches back to the previous page. The currently visible page is called active page.



Additionally, the ‘*Navigate*’ menu contains several helpful commands to navigate in your project and to keep the overview (see “[Navigate Menu](#)” (Chapter 1.5)).

Generally, the Composer supports drag&drop, copy&paste and the context menu, which appears after a click with the right mouse button. For example, to add a new unit to a project, drag a unit template with the mouse from the Template view (see “[Gallery](#)” (Chapter 2)) and drop it into the project page of the Composer. The new unit will be displayed as a brick at the same location where you dropped it.

If, while you are dragging an object, the desired page is not active, you can move the mouse to the corresponding tab in the Navigation bar and wait about half a second, then the Composer activates the corresponding page, where you can drop the object.

Within the Composer the members can be selected in several ways: A single member can be selected by clicking on it with the left mouse button. By clicking on a member while holding down the *Shift* key, the clicked member will be added to the selection or if the member is already selected, it will be removed from the selection.

To select several members you can drag a selection frame while the left mouse button is hold down. After changing the current selection in the Composer, the selection within the Inspector (see “[Inspector](#)” (Chapter 4)) is synchronized and vice versa.

To select all members of Composer use the command ‘*Select all*’ from the main menu ‘*Edit*’. If you wish to cancel the selection press the key *Esc*.

You can select a member also during a drag&drop operation in order, for example, to be able to drop the dragged content into the selected member. To do this, move the mouse cursor over the member and wait a short time until the member becomes selected. Now you can see its properties and attributes in the Inspector and you can drop there the dragged content.

If you wait a little bit longer, the content of the member will be opened. For example, you can drag something over a unit brick and wait until the unit is opened in a new Composer page. Now you can drop the dragged content directly into the opened unit.

The context menu of the Composer contains commands for ‘Undo & Redo’ for ‘Copy & Paste’ and for restacking selected objects. These commands are described in “[Edit Menu](#)” (chapter 1.2).

 Open	Return
 Undo	Ctrl+Z
 Redo	Ctrl+Y
 Cut	Ctrl+X
 Copy	Ctrl+C
 Paste	Ctrl+V
 Delete	Del
Select all	Ctrl+A
Rename member ...	F2
 Restack up	Ctrl+Up
 Restack down	Ctrl+Down
 Restack top most	Ctrl+Shift+Up
 Restack back most	Ctrl+Shift+Down

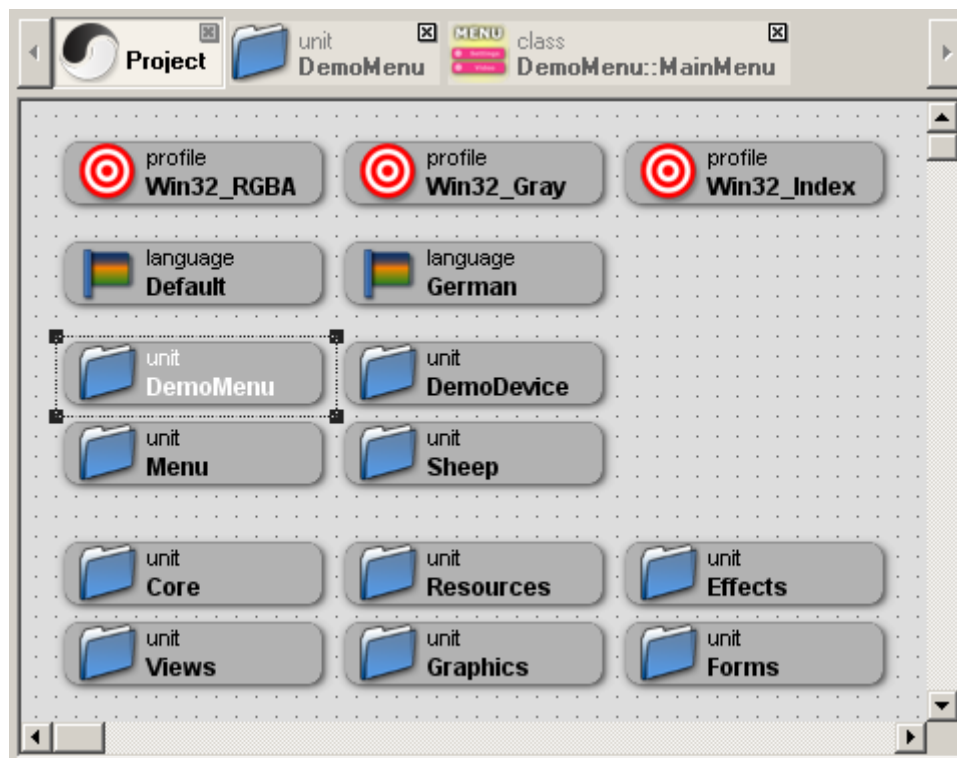
The appearance of the Composer page and the editing possibilities depends on the currently edited content. Thus, Embedded Wizard contains the following types of Composer pages:

- [Project Composer](#) (Chapter 3.1)

- [Profile Composer](#) (Chapter 3.2)
- [Unit Composer](#) (Chapter 3.3)
- [Enum Composer](#) (Chapter 3.4)
- [Set Composer](#) (Chapter 3.5)
- [Class Composer](#) (Chapter 3.6)

3.1 Project Composer

The content of the project page within the Composer reflects the content of the currently opened project (the *.ewp file). In contrast to all other Composer pages, the project page cannot be closed. The project page is closed automatically, when the project itself is closed. The following screenshot shows the Composer with an active project page:



A project page of the Composer may contains following members:

	Represents a single unit used in the project. The entire Embedded Wizard project consists of units. The units in turn store the project components, its classes, resources, etc. A double click on a unit brick opens a new Composer page containing the unit members.
	A profile describes the settings for the target platform. The target platform is determined by the name of the so-called Platform Package. Besides the name of the Platform Package used, a profile

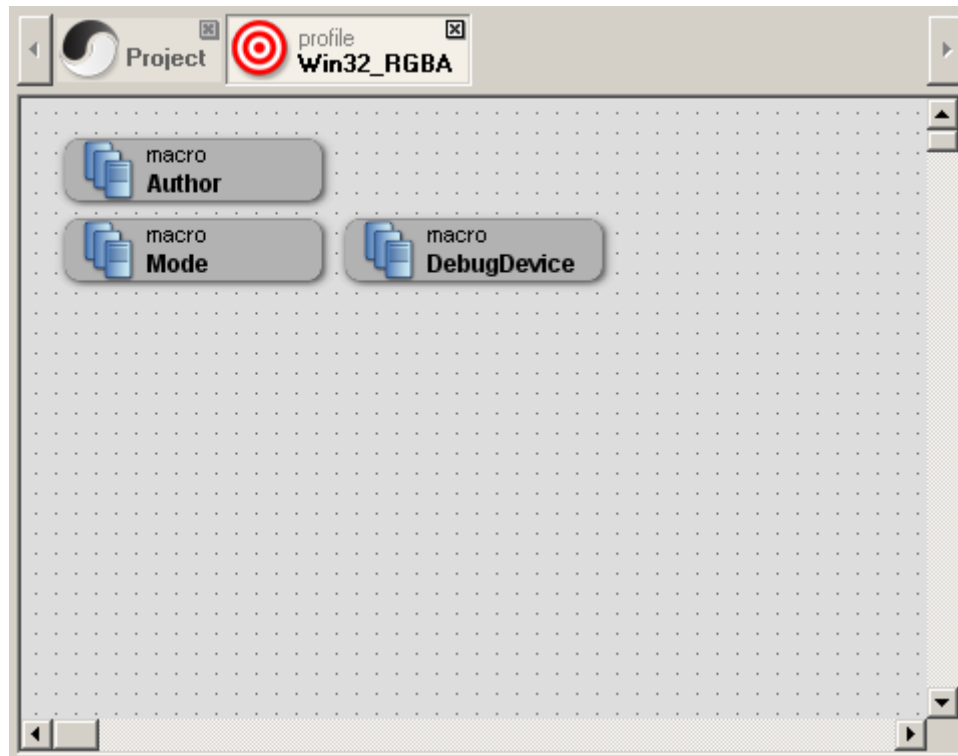
	may contain a set of settings and macro definitions. A double click on a profile brick opens a new Composer page containing the profile macros. Each project has to contain at least one profile.
	Define a supported language for which the GUI is localized. The localization makes it possible to develop an GUI in German, English, Japanese and so on. Each project has to contain at least the Default language.
	Styles are used to identify different variants of the GUI application. Such multivariant GUI application may change its appearance and behaviour at the runtime. This feature provides a simple way for the development of GUI applications with different skins and themes.

For details concerning the project and its members, please see the “Chora User Manual”

3.2 Profile Composer

The Profile Composer shows all macros of the currently edited profile. A profile may consist of macros only. A single macro stores any character string under a unique name. By using this name, the content of the macro (the character string stored there) can be used in definitions of classes, methods, and so on.

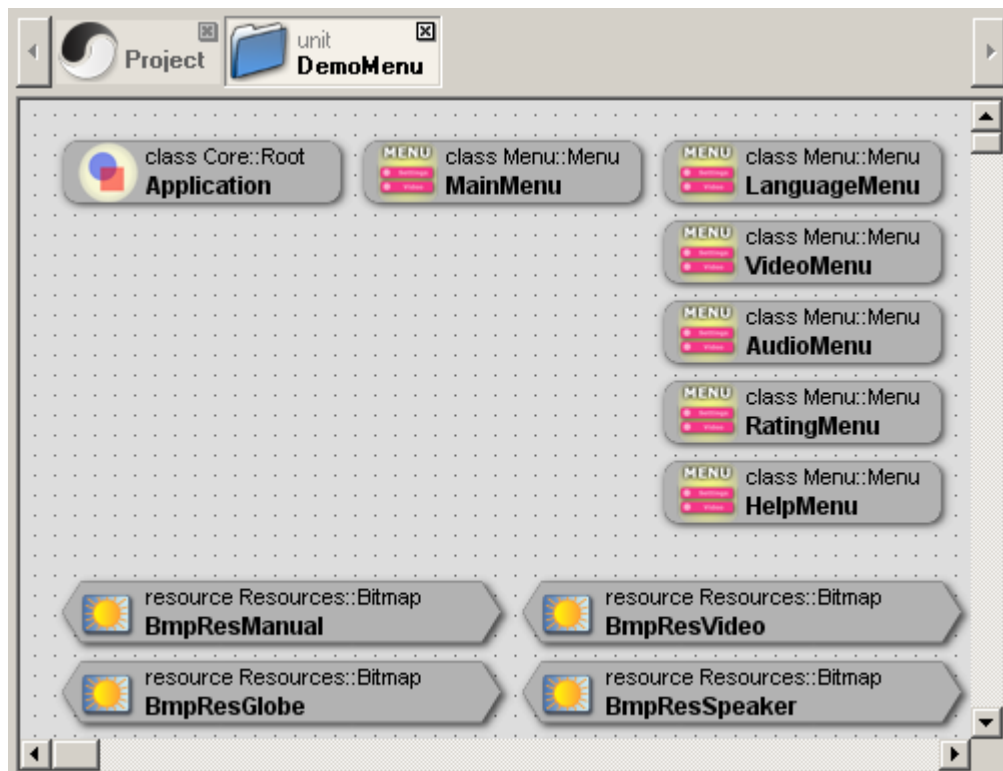
A macro is always defined within a profile; thus, a macro is valid only when the corresponding profile has been selected (see “[Build Menu](#)” chapter 1.6). The following screenshot shows the Composer with an active profile page:



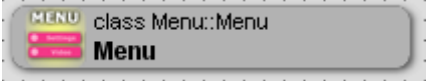
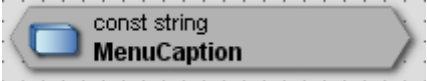
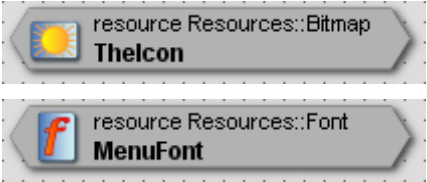
For details concerning profiles and macros, please see the “Chora User Manual”.

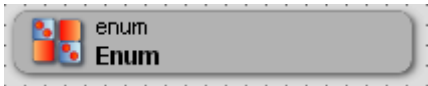
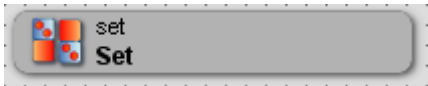
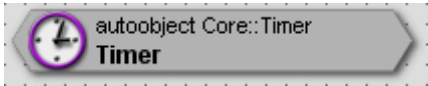
3.3 Unit Composer

The Unit Composer displays all definitions stored within the currently edited unit. The following screenshot shows the Composer with an active Composer page of the unit DemoMenu:



A unit page of the Composer may contains following members:

	The class. The class represents an object-oriented component of the GUI application. It consists of variables and methods. Classes can be derived, by what the derived classes inherit the members of their base classes. These inherit members can also be overridden and modified in derived classes. The icon displayed in the brick corresponds to the class itself or to the base class from which it has been derived. For user defined classes new icons can be copied into the <code>Icons</code> directory of Embedded Wizard. Beside the icon, the brick also displays the name of the base class. A double click on a class brick opens a new Composer page containing the implementation of the class.
	A constant bears a name and has a fixed value defined by the programmer. Using this name, the value of a constant can be used in the GUI application. Constants are excellently suited to defining, for example, menu titles or the colors of an GUI. Changing a constant, therefore, affects the entire GUI. The data type of the constant is displayed in the brick.
	Resources are constant files that the Chora compiler adopts in the ANSI C code generated. Thus it is possible, for example, to embed bitmaps in the GUI in order to use them at runtime from the GUI and display them on the screen. Embedded Wizard supports two kinds of resources: Bitmap resources and Font resources.
	Inline is a special definition that makes the Chora compiler insert a native code sequence in the generated ANSI C code. In this manner, the combination of Chora and ANSI C code is possible. This technique is used primarily in developing driver classes geared to specific platforms. Such driver classes form the interface between the GUI and the underlying target system, and guarantee the platform independence of the GUI that has been

	developed. A double click on a inline brick opens the implementation of the brick in the Editor.
	An enumeration is a user-defined data type that consists of a list of constant labels, so-called enumerators. A variable of the type of such an enumeration may, therefore, take on exactly one value from the enumerator list. A double click on an enum brick opens a new Composer page containing the enumerator items of the enum.
	A set is a user-defined data type that consists of a list of constant labels, so-called enumerators. A variable of the type of such a set may, therefore, take any combination of the enumerators – even a set. A double click on an set brick opens a new Composer page containing the enumerator items of the set.
	Objects, which are automatically instantiated on-request and released as soon as they are not in use anymore. Auto objects always exist in the global scope of a unit so they can be accessed from any part of the Chora code. The icon displayed in the brick corresponds to the class of the object. Beside the icon, the object also displays the name of its class. A double click on a object brick opens a new Composer page containing the implementation of its class.
	Variants of classes, constants, resources and objects. The variants are signed with a small . The concept of variants allows the development of GUI applications, with multiple different appearances and behaviors. The selection of the desired appearance and behavior is done during the code generation or at the runtime. The idea behind the variants is based on the object-oriented concept of derivation. This is realized in a way similar to the class inheritance – the affected parts of the GUI can be simple derived and adapted. The one big difference is, that variants do override (substitute) the origin definitions they are derived from. If at the runtime the definition is used,

	Embedded Wizard does handle with the variant instead.
--	---

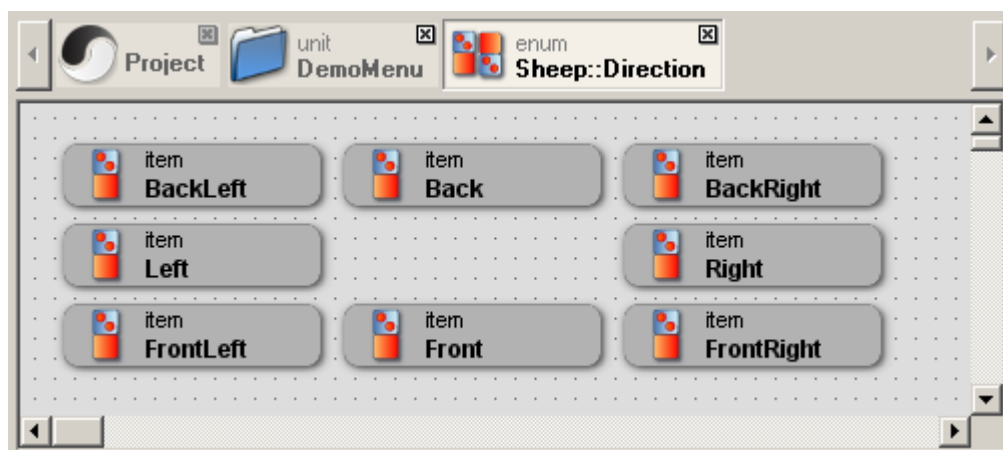
While working with unit members, you will frequently need to derive new classes, variants or objects:

- To derive a new sub-class from another class, drag&drop the base class brick by holding the *Ctrl+Shift* keys down.
- To derive a new variant from a class, resource, constant or an auto object, the keys *Ctrl+Shift+Alt* should be hold down during the drag&drop operation.
- To instantiate an auto object of a class, the keys *Ctrl+Alt* should be hold down while dragging the class brick.

For details concerning the unit, please see the “Chora User Manual”

3.4 Enum Composer

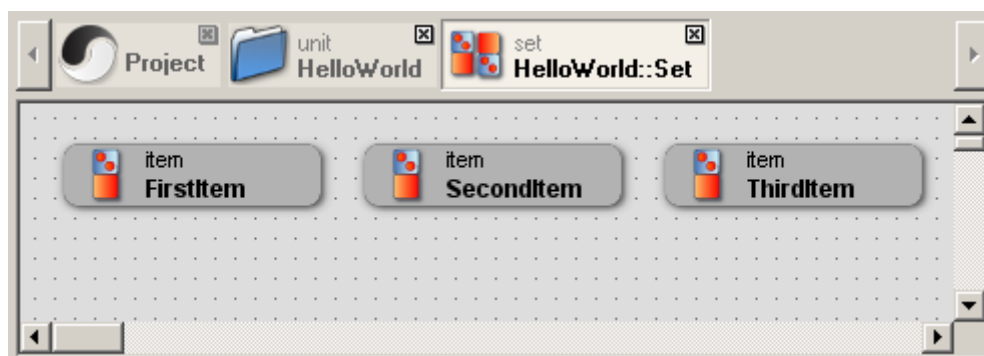
The Enum Composer is used to define all enumerators within an enumeration. The following screenshot shows the Composer with an active enum page:



For details concerning the enumeration, please see the “Chora User Manual”

3.5 Set Composer

The Set Composer is used to define all enumerators within a set. The following screenshot shows the Composer with an active set page:



For details concerning the sets, please see the “Chora User Manual”

3.6 Class Composer

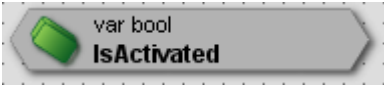
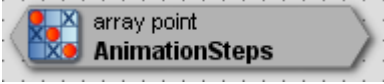
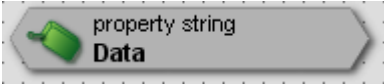
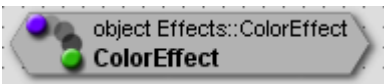
The Class Composer displays the content of a class. The appearance of the Class Composer depends on the origin of the class:

- If the class is derived from the Mosaic class `Core::Group`, the Composer contains an additional display area (so called ‘canvas’) surrounded by a thick blue frame. All graphical objects of the class (like rectangles, images, texts, ...) are displayed inside this display area in their real form of appearance. In this area it is also possible to select, resize and move the members with the mouse. All non graphical members of the class (like variables, properties, methods, ...) are drawn outside this display area as bricks.
- All other classes, which are not derived from `Core::Group`, are displayed only by using bricks. These classes do not have their own visual representation in the target system. They are used in the background of the GUI application, like drivers or containers for data, etc.

The following screenshot shows the Composer with an active class page. As you can see, the Composer also contains a canvas area, where the menu items, etc. are visible.



A class page of the Composer may contains following members:

	Variable that stores a value during the lifetime of an object, whereby each object contains its own copy of the variable. An object's variables store the state of that object, for example the color in which a shape is to be drawn on the screen. The data type of the variable is displayed in the brick.
	An array is a variable that can store more than one value. The values within the array can be accessed by using an index. The data type of the array is displayed in the brick.
	A property is an 'intelligent' variable. In a manner completely transparent to the programmer, accessing a property invokes a special onget or onset method, in which each property has its own methods. The programmer may define the logic of these methods himself in order to react to the specific aspects of read or write access to the property. For example, the programmer can have the screen contents updated automatically whenever a property is changed. The data type of the property is displayed in the brick.
	Any object may embed other objects of a different class. Each embedded object belongs to the superior object. Each superior object contains a copy of all the objects embedded within it. When an object is created at runtime, all objects embedded within it are automatically created as well. The icon displayed in the brick corresponds to the class of the object. Beside the icon, the object also displays the name of its class.
	A method is a function of an object. By calling the method, an object can be forced to perform a particular action.
 	The onget/onset methods are specialized methods which are invoked whenever the value of the corresponding property is to be queried or changed. In this manner the access to the property can be controlled.
	The slot method is a specialized method that can handle signals from other objects. Signals exist for the sake of simple communication between objects. If an object sends a signal to a particular slot of another object, the logic of the corresponding slot method of

	the recipient object is automatically executed.
--	---

Since a class can be derived from another class, it may consist of a lot of members. To keep the overview, only those members are displayed as bricks, which are added to or overridden in the class. All other inherited, but not overridden members are not displayed within the Composer. Of course, these members appear in the Inspector.

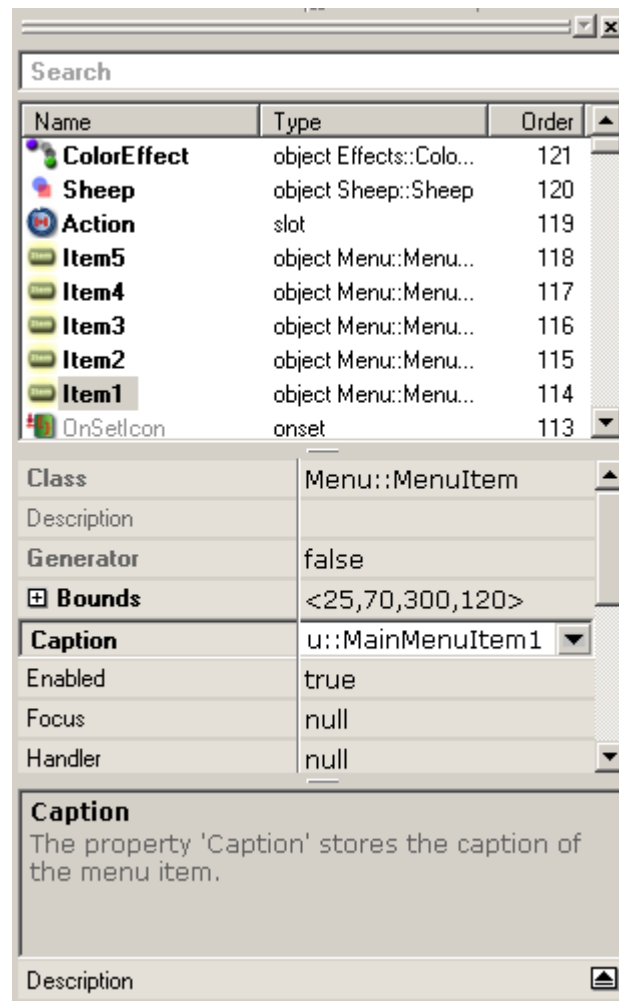
While working with the class Composer, you will frequently need to override class member or to embed objects of other classes:

- To override an inherited member (for example a method) you have to hold down the keys *Ctrl+Shift* and drag the member from the Inspector into the Composer.
- To restore to the inherited member, it is sufficient to delete the overridden member within the Composer.
- To instantiate an object of a class, the keys *Ctrl+Alt* should be hold down while dragging the class brick.

For details concerning classes, please see the “Chora User Manual”

4 Inspector

The Inspector of the Embedded Wizard is used to inspect and modify the members of the GUI application. The content of the Inspector corresponds to the content of the currently active page in the Composer (see “[Composer](#)” (Chapter 3)). Each time, the content within the Composer window changes, the content of the Inspector is updated. The following screenshot shows the Inspector:



The Inspector is divided into three parts:

- [Member Area](#) (Chapter 4.1)
- [Attribute and Property Area](#) (Chapter 4.2)
- [Description Area](#) (Chapter 4.3)

As long as the Inspector is the active window, you can press the keys *Ctrl+Tab* to toggle between Member Area and the Attribute/Property Area.

4.1 Member Area

The upper part of the Inspector contains a list of members. The content of the list corresponds to the content of the currently active page in the Composer. Each list entry corresponds to one member.

Name	Type	Order
 ColorEffect	object Effects::Colo...	121
 Sheep	object Sheep::Sheep	120
 Action	slot	119
 Item5	object Menu::Menu...	118
 Item4	object Menu::Menu...	117
 Item3	object Menu::Menu...	116
 Item2	object Menu::Menu...	115
 Item1	object Menu::Menu...	114
 OnSetIcon	onset	113

The list is divided in three columns:

- *Name* – shows the names of the members.
- *Type* – shows type of the members.
- *Order* – shows the z-order of the members. The z-order determines the drawing order of all visible objects from back to top. Additionally, the z-order determines the initialization order of the members during the runtime.

To sort a list by a column, click with the left mouse button on its caption. For example, to sort the list by the type, click on the caption of the *Type* column. Please note, unlike other columns, the *Name* column provides two different sorting algorithms. If you click on the caption of the *Name* column, you can toggle between sorting by name and sorting by inheritance. In case of the sorting by inheritance, the Inspector will show first new members followed by inherited and overridden members and at the end the inherited but no overridden members.

Per default all members in the Member Area are displayed with black text color. Only class members are signed with different text colors and fonts in order to reflect their origin and the class dependencies.

 HandleEvent	All new defined, not inherited members are drawn bold with the black text color.
 HandleEvent	Inherited and overridden members will be drawn bold with the gray text color.
 HandleEvent	Inherited but not overridden members will be displayed with the gray text color and thin font.
 HandleEvent	In case of errors in the inheritance of the particular member, the member is drawn in red color. Such errors are detailed reported in the Log Window.

The list entries can be selected with the keyboard (analog to Windows Explorer) or with the mouse: A single click on a name selects the member. By holding down the *Ctrl* key, more than one member can be selected. A second click to an already selected member removes it from the selection. After changing the current selection in the Inspector, the selection within the Composer is synchronized and vice versa.

To select all members of Composer use the command '*Select all*' from the main menu '*Edit*'. If you wish to cancel the selection press the key *Esc*.

You can select a member also during a drag&drop operation in order, for example, to be able to drop the dragged content into the selected member. To do this, move the mouse cursor over the member and wait a short time until the member becomes selected. Now you can see its properties and attributes in the middle part of the Inspector and you can drop there the dragged content.

If you wait a little bit longer, the content of the member will be opened. For example, you can drag something over a unit member and wait until the unit is opened in a new Composer page. Now you can drop the dragged content directly into the opened unit.

When you click with the right mouse button, a context menu of the Member Area appears. The context menu contains commands for 'Undo & Redo' for 'Copy & Paste' and for restacking selected objects.

Open	Return
Undo	Ctrl+Z
Redo	Ctrl+Y
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Delete	Del
Select all	Ctrl+A
Rename member ...	F2
Override class member ...	Ctrl+M
Restack up	Ctrl+Up
Restack down	Ctrl+Down
Restack top most	Ctrl+Shift+Up
Restack back most	Ctrl+Shift+Down

Furthermore the command 'Open' opens the currently selected member in Composer or Editor (depending on type des members).

With the command 'Rename member' the currently selected member can be renamed. Please note, that you cannot rename inherited, still not overridden members - their name is already determined by the inheritance. When you select this command, a small inplace edit field appears, where the new name of the member can be entered. To apply the new name and to close the inplace edit field press the key *Return*.

Name	Type	Order	
 MoveView	method	13	
 Draw	method	12	
 InvalidateView	method	11	
 InvalidateRect	method	10	
 IsExposed	method	9	
 OnSetMode	onset	8	
 OnSetVisible	onset	7	
 Mode	property Graphics:...	6	

When renaming a member, Embedded Wizard evaluates the entire project and searches for references to the renamed member in order to adapt them to the new name. This, so called, 'refactoring' take in account the inheritance of classes, the initialization expressions of members, the source code of methods, etc. To suppress the automatic refactoring, press the keys *Shift+Return* instead of the key *Return* when closing the inplace edit field.

The command '*Override class member ...*' is useful to override inherited members. For example, if you like to override an inherited method, select the method in the Inspector and apply the command '*Override class member ...*'. Thereupon the method will become overridden → a method brick appears in the Composer. Now you can open the overridden method for editing, etc.

The edit commands are described more in detail in "[Edit Menu](#)" (chapter 1.2).

The upper area of the Inspector provides a 'Search' edit field. This edit field can take a filter condition. If the filter is set, the Inspector will show only the members, whose name does fit the given filter condition:



The condition is fulfilled, if the filter text does appear anywhere within the members name. The filter evaluation is case insensitive. For example, if the filter is set to *event*, the members *HandleEvent* and *DispatchEvent*, etc. are displayed.

Optionally, parts of the filter can be specified using the wildcards '*' and '?'. The sign '*' stands for an arbitrary number of any signs. The '?' wildcard, on the other hand, stands for exactly one sign. When using wildcards, the entire name of the member has to fit the filter condition otherwise it will be excluded. To filter all members, set the filter to a single * sign.

To deactivate the filter click on the icon  on the right hand side of the edit field or press the *Esc* key as long as the edit field is focused.



As long as the Inspector is the active window, you can press the key *Tab* to toggle between the 'Search' edit field and the list of members.

4.2 Attribute and Property Area

The middle part of the Inspector contains the attributes and properties of the currently selected members. If only one member is selected, all attributes and properties of the member are displayed. If two or more members are selected, only the common attributes and common properties are displayed:

 Brick	<340,190,480,230>
Class	Effects::ColorEffect
Description	This is a pulse color ...
Generator	false
Actor	^Rectangle.Color
Begin	0
Enabled	true
 EndValue	#FFFFFF

The list is divided in two columns:

- Name – at the left the names of attributes and properties is shown.
- Value – at the right the values of attributes and properties is shown.

The both columns are divided by a thin splitter, on which you can click with the left mouse button in order to grab and drag it.

The colors used to display the attributes and properties reflect the state of the affected attribute or property:

Description	All attributes, which are not modified, are drawn with thin font and gray text color. These attributes are initialized with a value inherited from the base class, or they are initialized with default values.
Description	Modified attributes are displayed bold with the gray text color.
Enabled	Properties are always displayed in the black text color. Properties, which are not modified, are displayed with the thin font.
Enabled	Modified properties are displayed bold with the black text color.
Enabled	Attributes or properties signed with the red color are invalid. The appropriate error message is displayed in the Log window when the affected attribute is evaluated by the Prototyper or during the code generation.

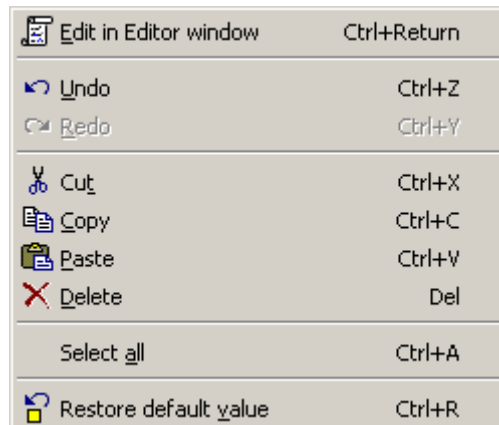
If an attribute or a property contains several subordinate values, a small  icon appears on the left hand side. By clicking on it, the attribute or the property can be expanded and you can inspect the subordinate values. For example, a color property can be expanded in order to inspect its red, green, blue and alpha components:

Generator	false
 Bounds	<100,170,200,270>
 Color	#0000FFFF
red	0
green	0
blue	255
alpha	255
Mode	Graphics::Mode.Blend

In case of the selected attribute or property, the Inspector provides an edit field right to its name. It can be used to modify the value of the selected attribute or property. Here you can enter initialization expressions for properties or you can specify attributes of class members, etc.

 Color	#0000FFFF
---	-----------

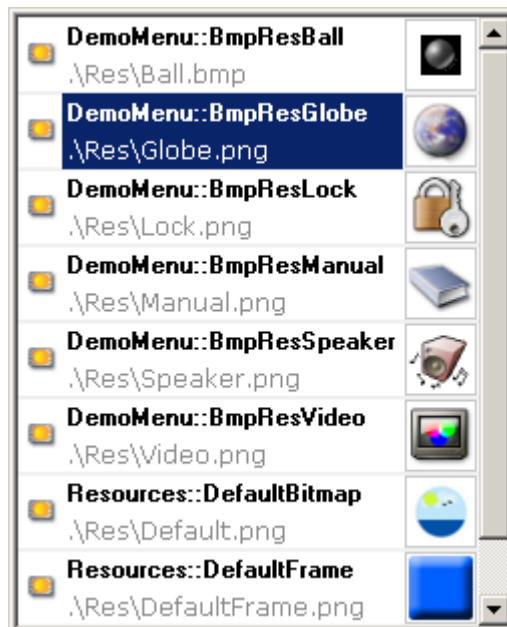
By clicking with the right mouse button on an attribute or property, a context menu appears. From this menu you can select several commands:



The command 'Edit in Editor window' allows you, for example, to connect the edit field of the Inspector with the Editor (see "Editor" (Chapter 5)). In this manner you can enter complex expression or long text more comfortable directly into the Editor instead of the small edit field. At the end of the editing press the key *Ctrl+Return* to transfer the data from the Editor into the Inspector.

If you want to restore the previously modified attribute or property to its origin, inherited value, use the command 'Restore default value' from the context menu.

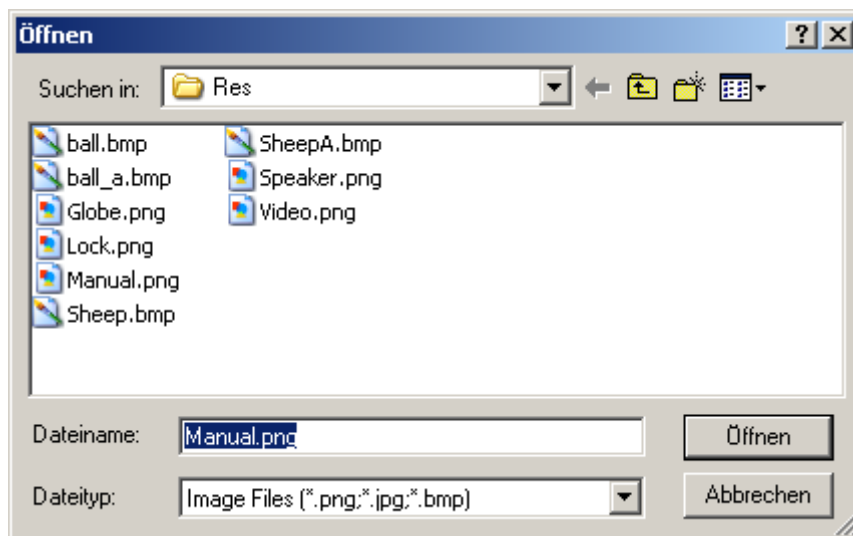
For your convenience the Inspector also provides a reach set of so called Assistants. By using them you can modify the attributes or properties of selected members much faster. For example the Resource Assistant allows you to determine a bitmap, which should be displayed in front of a menu item by selecting it from a list of existing bitmaps of your project.



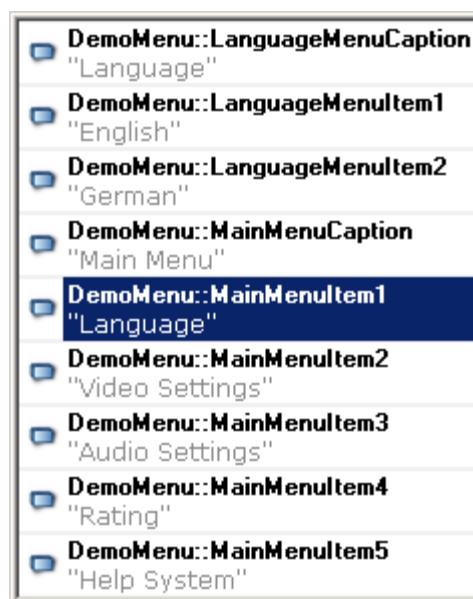
To open an assistant click on the small button ▼ or ... on the right hand side of the selected attribute or property or press the *Return* key. The kind of the displayed assistant corresponds to the data type of the selected attribute or property. If there is no assistant available, no button is displayed. We distinguish between three groups of assistants:

- Dialog assistants
- Popup list assistants
- Color assistant

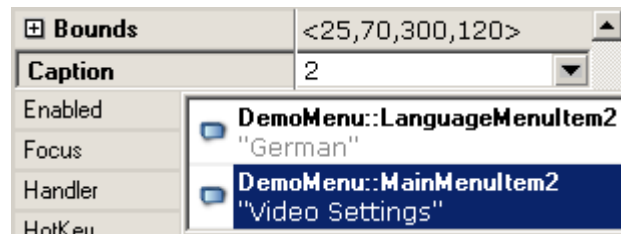
The dialog assistants include the 'File Open' dialog and the 'Select Directory' dialog. They appear, for example, when you are editing the *FileName* attribute of a bitmap resource and you wish to select the desired bitmap file.



Popup list assistants display a list of entries, which fit to the edited attribute or property. For example, if you are editing a *string* property, the assistant will provide you with the list of all *string* constants of your project. In this manner you can select the special one and apply it by pressing the *Return* key:

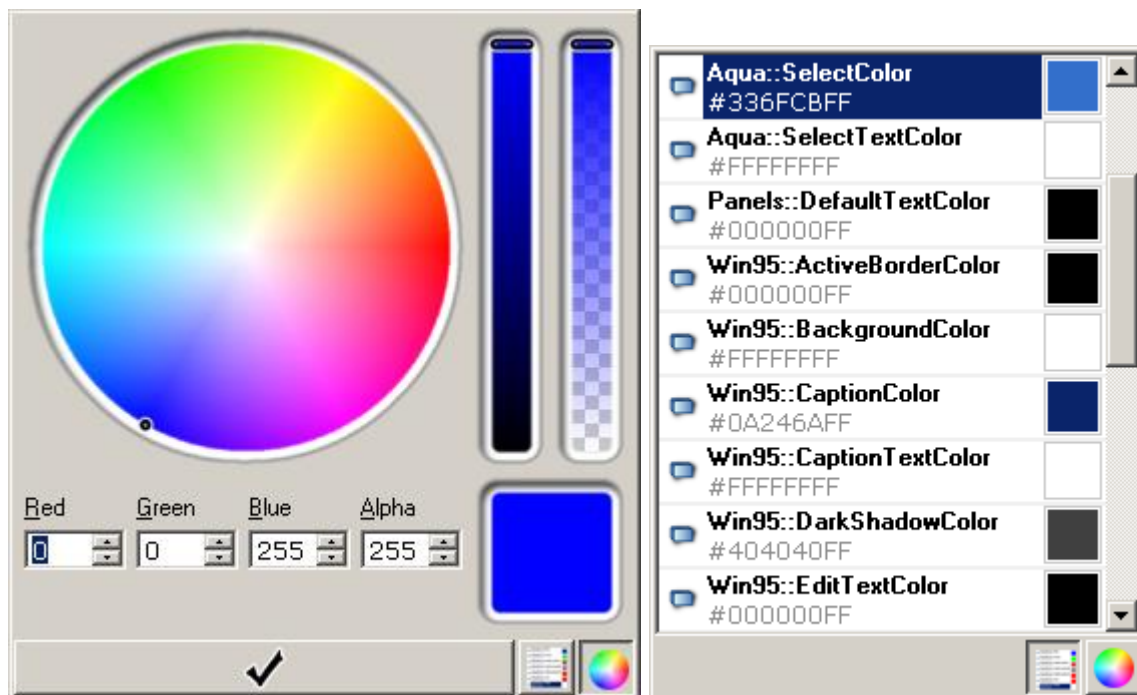


As long as a popup list assistant is visible, you can apply a search filter to the displayed entries in order to find the desired one in a more convenient way. To do this simply input a part of its name. The entered filter is displayed directly in the edit field of the Inspector. Thereupon the assistant shows only these members, whose names do contain the given search filter and you can select the desired one much faster:



To cancel the previously entered filter, press the *Esc* key – so the complete list of entries is displayed again. With another *Esc* key the assistant itself will be closed and you can continue working with the Inspector. An alternative way to close the popup assistant is to click with the mouse outside the assistant. In this case the assistant becomes deactivated and closes.

The color assistant is a special kind of the popup list assistant. It provides two views: the list view with all available color constants of the project and the color wheel view, where the desired color can be directly determined in a very convenient way.



On the bottom of the color assistant you can see several buttons. With the both buttons   you can switch between the list view and the color wheel view. This can also be done by pressing the keys *Ctrl+Tab*.

The  button appears only in the color wheel view and can be used to apply the selected color in a more convenient way. The same can be reached by pressing the *Return* key.

Please note, if your project does not contain any color constants, the color assistant is limited to the color wheel view only. In this case you cannot toggle between the both views.

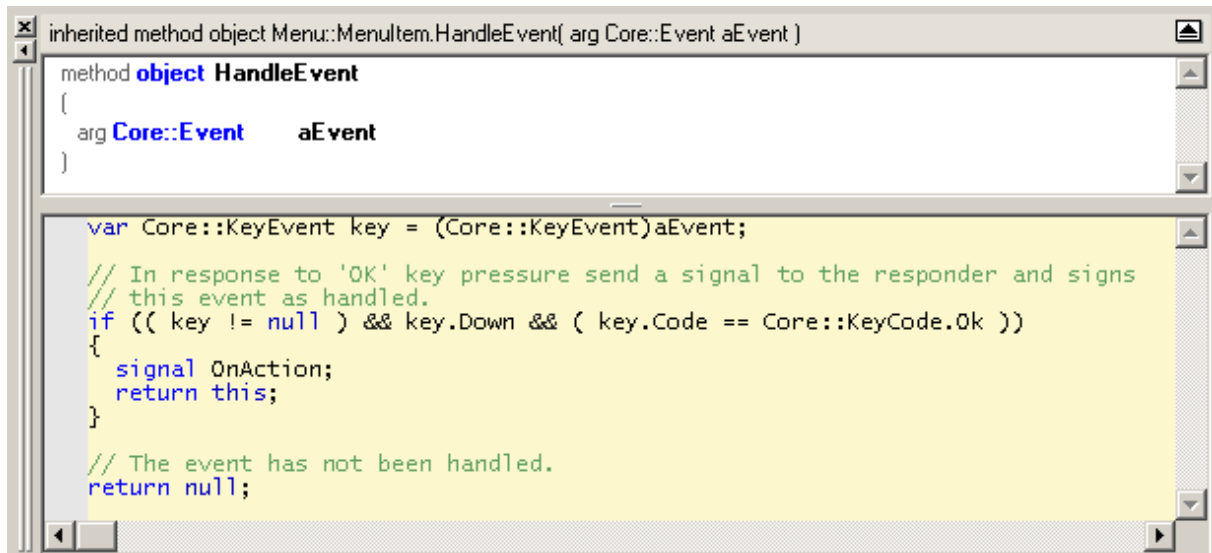
4.3 Description Area

The lower part of the Inspector displays the description of the currently selected member, attribute or property. To hide and show the description area click on the arrow  at the bottom of the Inspector. The following screenshot shows the Description Area of the Inspector:



5 Editor

The Editor of the Embedded Wizard is mainly used to implement methods. Additionally, the Editor can be used to insert complex expressions or long text in the Inspector more comfortable (see “[Attribute and Property Area](#)” (Chapter 4.2)). For this purpose, the Editor can be connected with the Inspector by selecting the command ‘*Connect Editor to Inspector*’ from the ‘*Window*’ main menu. The following image shows the Editor:



The Editor is divided into two parts:

- [Declaration Area](#) (Chapter 5.1)
- [Implementation Area](#) (Chapter 5.2)

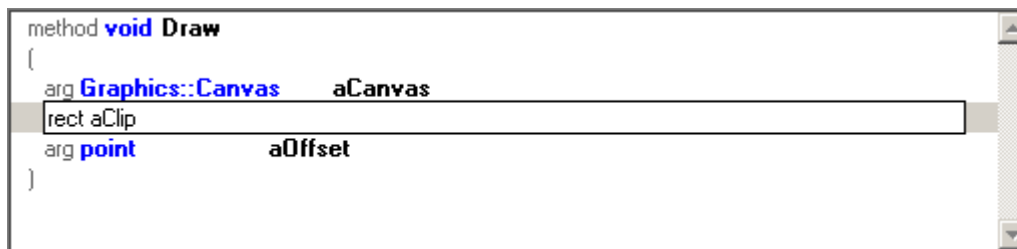
To drop down the Declaration Area click on the small  button on the right hand side of the Editor. The same can be reached when you press the *Ctrl+Tab* key. If the Declaration Area is already dropped, you can use *Ctrl+Tab* to toggle between these both views.

5.1 Declaration Area

The upper part of the Editor displays the declaration of a method and can be dropped down to inspect and change the return type of the method, to add new parameters, etc. The following screenshot shows the Declaration Area within the Editor:



To edit an entry you can double click on it or press *Return* or *F2* when an entry is selected. When you select this command, a small inplace edit field appears, where the name and the type of the parameter can be entered. To apply the changes and to close the inplace edit field press the key *Return*.



When renaming a parameter, Embedded Wizard evaluates the body of the method and the body of all derived methods in order to adapt all expressions, where the parameter has been used. This, so called, 'refactoring' take in account the inheritance of classes. To suppress the automatic refactoring, press the keys *Shift+Return* instead of the key *Return* when closing the inplace edit field.

To add a new parameter, you can press *Ins* key or double click on one of the parentheses '(' or ')'. Alternatively, you can drag the *Argument* template from the *Default* folder of the Gallery to add a new parameter.

If you want to delete an argument, select it and press the *Del* key. If you delete the return value of a method, the *void* return type is assumed.

Please note, that the declaration of a derived method cannot be changed. In this case all editing commands are disabled. Only the origin definition of a method allows the modification of the declaration. Derived methods do inherit this declaration only.

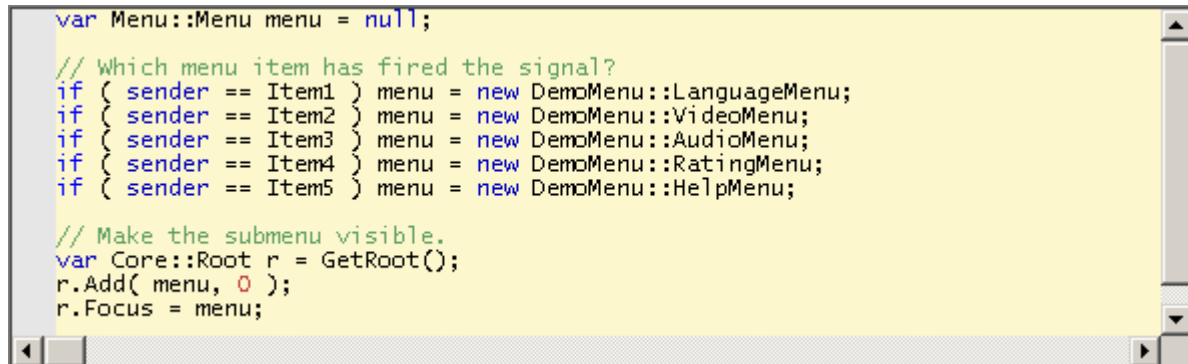
When clicking with the right mouse button on the Declaration Area, a context menu appears. It provides the commands for 'Undo & Redo' for 'Copy & Paste', for inserting a new argument and for restacking the arguments.

Edit ...	Return
Undo	Ctrl+Z
Redo	Ctrl+Y
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Delete	Del
Insert new argument ...	Ins
Restack up	Ctrl+Up
Restack down	Ctrl+Down
Restack top most	Ctrl+Shift+Up
Restack back most	Ctrl+Shift+Down

The reordering of parameters can also be done more comfortable with the mouse. To do this select the affected parameter and drag&drop it to its new position.

5.2 Implementation Area

The lower part of the Editor displays the implementation of the method. Here you can program with Chora statements. The following screenshot shows the Implementation Area within the Editor:



```

var Menu::Menu menu = null;

// Which menu item has fired the signal?
if { sender == Item1 } menu = new DemoMenu::LanguageMenu;
if { sender == Item2 } menu = new DemoMenu::VideoMenu;
if { sender == Item3 } menu = new DemoMenu::AudioMenu;
if { sender == Item4 } menu = new DemoMenu::RatingMenu;
if { sender == Item5 } menu = new DemoMenu::HelpMenu;

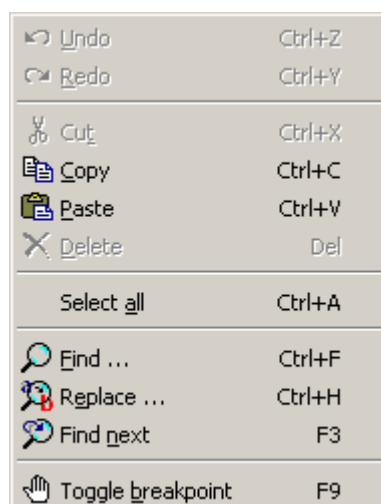
// Make the submenu visible.
var Core::Root r = GetRoot();
r.Add( menu, 0 );
r.Focus = menu;

```

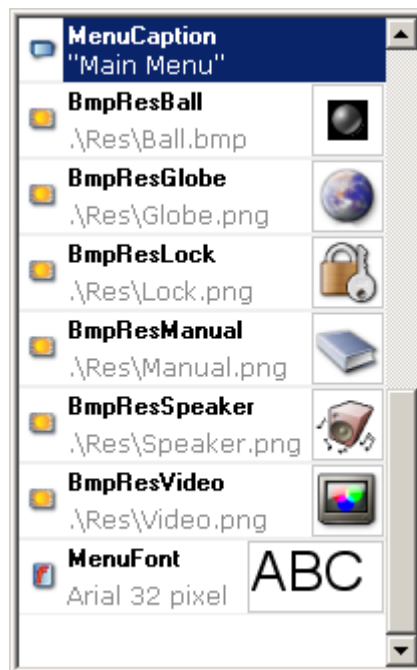
The Editor supports the following features:

- Syntax coloring for comments and Chora keywords
- 'Drag & Drop', 'Copy & Paste', 'Undo & Redo' and context menu
- Search (*Ctrl+F*), Find Next (*F3*) and Replace (*Ctrl+H*)
- Toggle Bookmark (*Ctrl+F2*), Go to Next Bookmark (*F2*) and Go to Previous Bookmark (*Shift+F2*)
- Display Whitespaces (*Ctrl+Shift+8*)
- Block Indent Right (*Tab*) and Block Indent Left (*Shift+Tab*)
- Toggle breakpoint (*F9*)

Some of the features can be activated in the context menu, which appears after a right mouse button click in the Implementation Area:

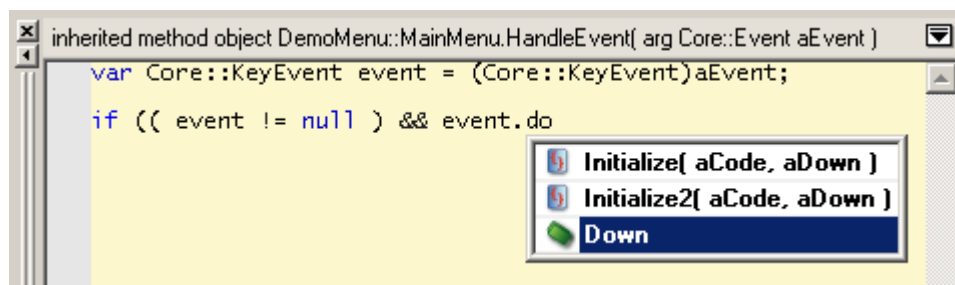


For your convenience the Editor also provides a set of so called Assistants. By using them you can enter expressions and select members much faster. For example the Scope Assistant displays a list of members, which are valid in scope of an edited expression:



This assistant appears automatically as soon as you have entered a period `.` sign and the expression left to this sign results in a valid object or a `string`, `color`, `rect`, etc. value.

As long as an assistant is visible, you can apply a search filter to the displayed entries in order to find the desired one in a more convenient way. To do this simply input a part of its name. The entered filter is displayed directly in the Editor. Thereupon the assistant shows only these members, whose names do contain the given search filter and you can select the desired one much faster:

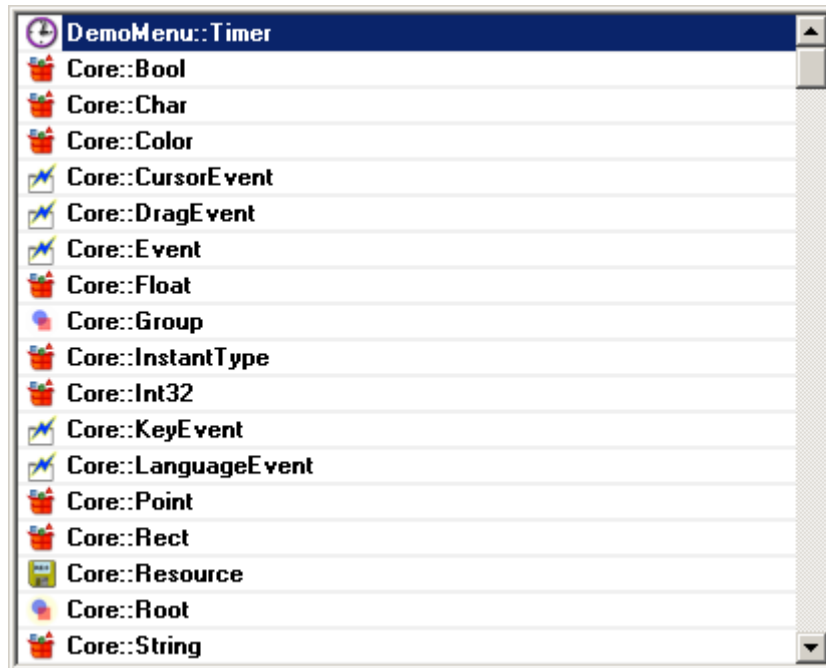


To cancel the previously entered filter, press the *Esc* key – so the complete list of entries is displayed again. With another *Esc* key the assistant itself will be closed and you can continue working with the Editor. An alternative way to close the assistant is to click with the mouse outside the assistant. In this case the assistant becomes deactivated and closes.

If you have entered a text while an assistant was visible and now you want to dismiss the assistant without deleting the entered text, simply move the caret outside the entered text by multiply pressing the keys *Left* or *Right*.

Beside the Scope Assistant, the Editor also provides other assistants. For example, the Macro Assistant witch appears after the entering the `$` sign. Or the Attributed Text Assistant, which appears, when the user has entered a brace `{` sign within a `string` literal.

By pressing the keys *Ctrl+Space* you can open scope independently a global assistant, which displays all project members and the members of the currently edited class. In this manner you get access to all members at once:



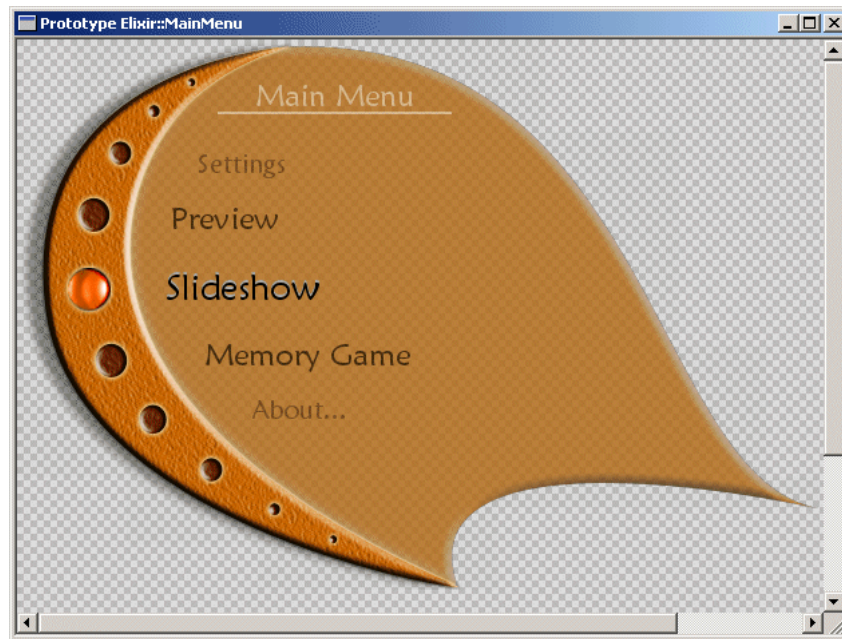
6 Prototyper

The Prototyper of the Embedded Wizard is used to test a single class or a complete GUI application. The Prototyper can be started by selecting the command 'Start Prototyper' from the 'Build' main menu or simply by pressing key *F5*. After starting the Prototyper, the class, which is currently edited or selected in the Composer (see "Composer" (Chapter 3)), is loaded into the Prototyper. The class and all embedded objects within the class are created and all graphical outputs are shown.

The Prototyper provides the environment with timer events and translates all keyboard events into Mosaic events. An overview with the supported keys is displayed after holding down the *Ctrl* key for few seconds:

Menu	ctrl M	Play	ctrl P	Audio	ctrl A
Up	↑	Stop	ctrl S	TV	ctrl W
Down	↓	Pause	ctrl ⇧ P	DVD	ctrl D
Left	←	Record	ctrl ⇧ R	VCR	ctrl V
Right	→	Rev	ctrl ←	EPG	ctrl E
Ok	Enter	Fwd	ctrl →	OSD	ctrl O
Exit	Esc	SlowRev	ctrl ⇧ ←	PIP	ctrl Q
Home	Pos1	SlowFwd	ctrl ⇧ →	Text	ctrl T
End	End	SkipBwd	ctrl ↑	Subtitle	ctrl U
Insert	Ins	SkipFwd	ctrl ↓	Index	ctrl X
Delete	Del	SkipPrev	ctrl ⇧ ↑	Info	ctrl I
Clear	←	SkipNext	ctrl ⇧ ↓	Zoom	ctrl Z
PageUp	Page ↑	Repeat	ctrl ⇧ E	Clock	ctrl C
PageDown	Page ↓	Angle	ctrl ⇧ A	Timer	ctrl ⇧ T
Show	ctrl Ins	Eject	ctrl ⇧ ←	Display	ctrl Pos1
Hide	ctrl Del	Game	ctrl ⇧ G	Navigation	ctrl ⇧ N
Red	ctrl R	Karaoke	ctrl ⇧ K	ChannelUp	ctrl ⇧ Page ↑
Green	ctrl G	Mode	ctrl ⇧ D	ChannelDown	ctrl ⇧ Page ↓
Blue	ctrl B	Mute	ctrl ⇧ M	VolumeUp	ctrl Page ↑
Yellow	ctrl Y	Setup	ctrl ⇧ U	VolumeDown	ctrl Page ↓
White	ctrl L	Power	ctrl End	Help	ctrl H
Magenta	ctrl N				
User0 - User9		ctrl 0 - 9		Key0 - Key9 0 - 9	
User10 - User19		ctrl ⇧ 0 - 9		F1 - F10 F1 - F10	

The following screenshot shows a running Prototyper with a GUI application:



Please note: If a GUI application remains in an endless loop, so that it can neither be controlled nor you can interact with the Embedded Wizard IDE, there are two ways how you can obtain the control again:

- On the keyboard you can press and hold down the key *Pause*.
- With the mouse you can click on the close button in the right corner of the Prototyper window.

In both cases the Prototyper is held and the current code line is shown in the Editor. Now you can investigate the reason for the endless loop using the Embedded Wizard integrated Debugger.

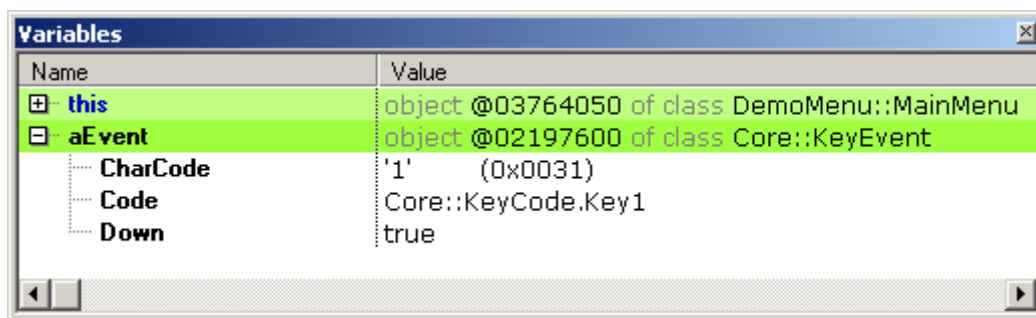
Clicking again the close button while the interpreter is held, will close the Prototyper window and terminate the prototyping.

7 Variables Window

The Variables window belongs to the group of monitor windows. These windows can be used while debugging the GUI application in order to inspect variables, objects, etc.

Unlike other monitor windows, Variables window do appear automatically when the Prototyper execution is halted. To see the other monitor windows, you have to activate them explicitly in the '*Window*' menu (see "[Window Menu](#)" (Chapter 1.9)).

While debugging the GUI application step by step, the Variables window evaluates the Chora statements and displays the operands, which are used in the last and the current statement. Operands used in other statements are not displayed. The following screenshot shows the Variables window:

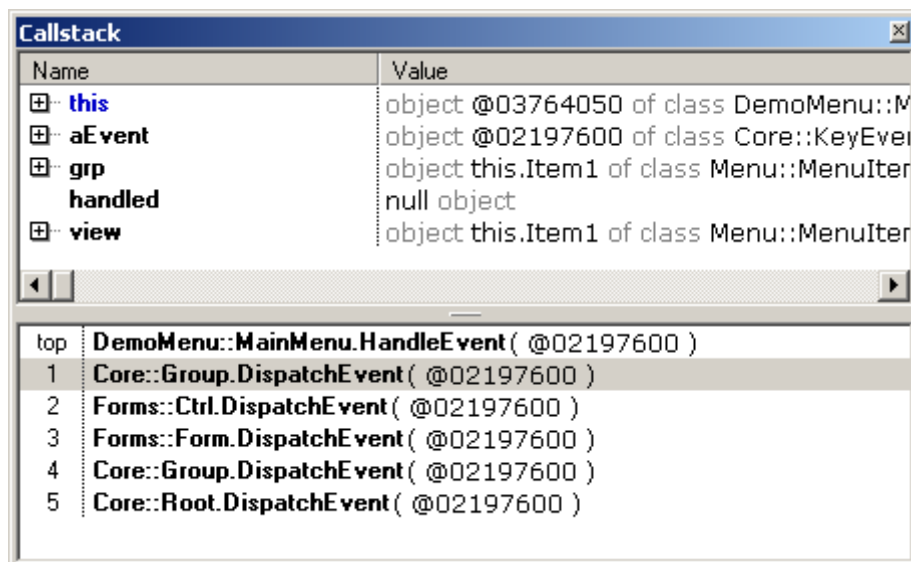


All monitor windows are always floating and they cannot be docked to the Main Window. To hide all monitor windows temporary use the key *F6*.

8 Callstack Window

The Callstack window belongs to the group of monitor windows. These windows can be used while debugging the GUI application in order to inspect variables, objects, etc.

The Callstack window allows you to inspect the nesting method calls. It is divided in two parts:



- In the lower part the list of currently executed methods is displayed. The entry on the top of the list corresponds to the lastly invoked method. The methods are displayed with the originally passed arguments as the method was invoked. A double click on a method will guide you to the corresponding statement in the Editor (see “[Editor](#)” (Chapter 5)).
- The upper part of the window displays the call frame of the method selected in the lower part of the window. Here are all local variables, the arguments and the ‘this’ object displayed.

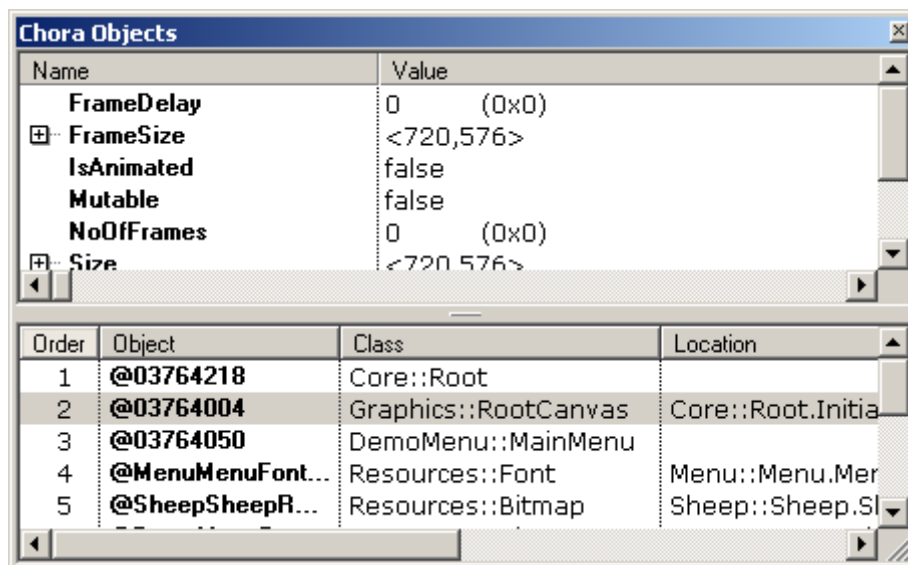
To activate the window, select the command ‘*Callstack*’ in the main menu ‘*Window*’ (see “[Window Menu](#)” (Chapter 1.9)).

All monitor windows are always floating and they cannot be docked to the Main Window. To hide all monitor windows temporary use the key *F6*.

9 Chora Objects Window

The Chora Object window belongs to the group of monitor windows. These windows can be used while debugging the GUI application in order to inspect variables, objects, etc.

The Chora Objects window allows you to inspect all existing Chora objects. It is divided in two parts:



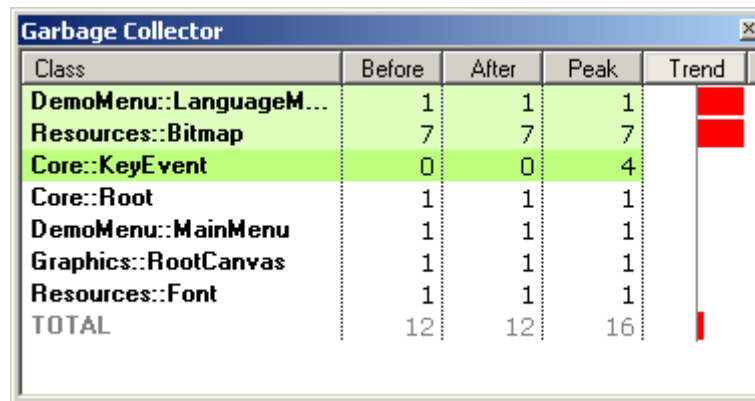
- In the lower part the list of dynamically created Chora objects is displayed. In the Prototyping environment the dynamically created objects are prefixed with the @ (at) sign followed by a unique id number of the object. In case of resource objects, the name of the resource followed by the language variant is used instead. A double click on an object will guide you to the statement, where the object has been created.
- The upper part of the window displays the content of the object selected in the lower part of the window.

To activate the window, select the command 'Chora Objects' in the main menu 'Window' (see "[Window Menu](#)" (Chapter 1.9)).

All monitor windows are always floating and they cannot be docked to the Main Window. To hide all monitor windows temporary use the key *F6*.

10 Garbage Collector Window

During the prototyping the Garbage Collector window can be displayed. It shows a brief statistic of the used objects, how many objects are alive and what happened during the last cycle of the garbage collection.



Class	Before	After	Peak	Trend
DemoMenu::LanguageM...	1	1	1	
Resources::Bitmap	7	7	7	
Core::KeyEvent	0	0	4	
Core::Root	1	1	1	
DemoMenu::MainMenu	1	1	1	
Graphics::RootCanvas	1	1	1	
Resources::Font	1	1	1	
TOTAL	12	12	16	

The column '*Before*' displays the number of objects *before* the last garbage collection. The column '*After*' displays how many objects have survived the garbage collection. The number of disposed objects results in the difference '*Before* – *After*'. The column '*Peak*' displays the maximal number of objects existing at the same time since the Prototyper was started.

The Garbage Collector window analyses the last 32 garbage collection cycles and calculates the trend, whether the number of objects increases, decreases or stays unchanged. This is displayed in the column '*Trend*'. Red bars mean that the number of corresponding objects increases. Green bars imply that the number of objects downsizes. The size of the bar indicates the variance.

Unlike other monitor windows, the Garbage Collector window is always available even the GUI application is not halted, so you can interact with the GUI application and observe how many objects are created and disposed. Via double click on a specific entry the Prototyper is stopped and the Chora Objects window is opened for inspecting the objects of the affected class. For further information see "[Chora Objects Window](#)" (Chapter 9).

To activate the window, select the command '*Garbage Collector*' in the main menu '*Window*' (see "[Window Menu](#)" (Chapter 1.9)).

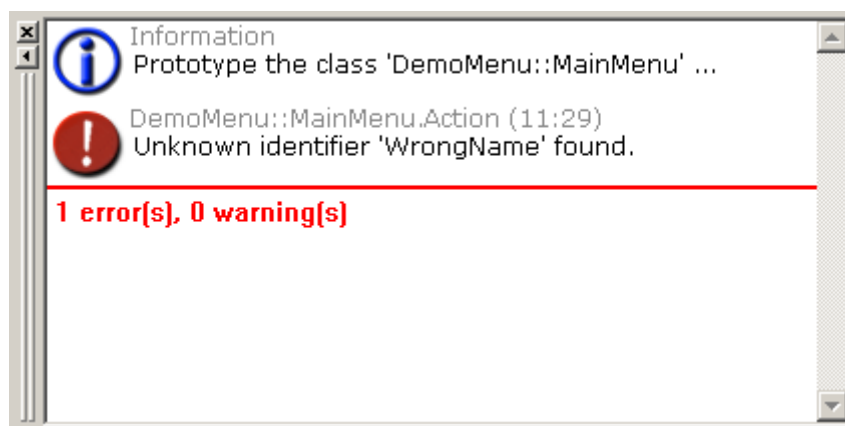
All monitor windows are always floating and they cannot be docked to the Main Window. To hide all monitor windows temporary use the key *F6*.

11 Log Window

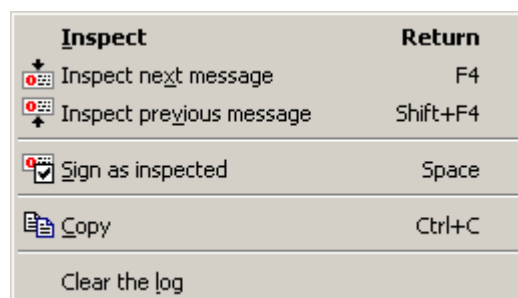
The Log Window of the Embedded Wizard displays short status reports and all warnings and errors occurred during compilation and prototyping. A double click on an error message will guide you to the error location: Embedded Wizard opens the affected definition and displays the erroneous location in the Composer (see “Composer” (Chapter 3)), Editor (see “Editor” (Chapter 5)) or in the Inspector (see “Inspector” (Chapter 4)).

Additionally, all search results of the command ‘*Find in project...*’ and all `trace` outputs are collected in the Log Window. More about the `trace` statement can be found in “Chora User Manual”.

The following screenshot shows the Log Window:



When clicking with the right mouse button on the Log Window, a context menu appears:



The Log Window provides following commands.

- *Inspect* – inspects the currently selected log message. If possible shows the origin of the message in the Composer, Editor or Inspector.
- *Inspect next message* – continue with inspecting of the next following message. The shortcut F4 is very useful if you want to traverse step by step through a large list of messages.
- *Inspect previous message* – continue with inspecting of the previous message.
- *Sign As Inspected* – It toggles the ‘inspected’ state of the message. Usually messages, which have been inspected, appear a little bit grayed out. In this manner you can see, which messages still wait for the inspecting.

- *Copy* – This command copies the entire content of the Log Window as text into the clipboard. In this manner you can append it to an email or to your documentation.
- *Clear the log* – Deletes all messages from the Log Window.